

ATARI®
PERSONAL COMPUTER SYSTEM
OPERATING SYSTEM
User's Manual

=====

NOTICE

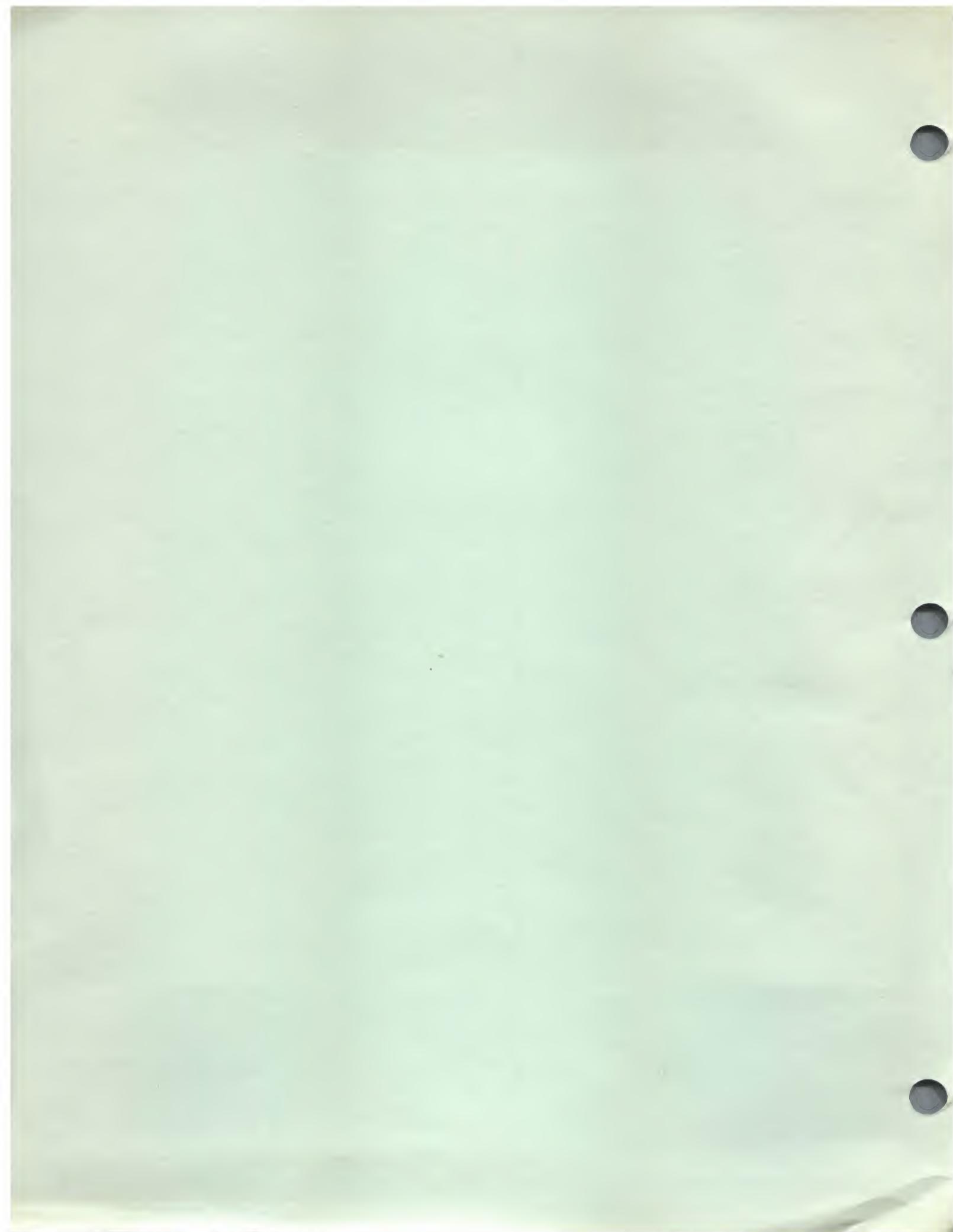
=====

TO ALL PERSONS RECEIVING THIS DOCUMENT

January 1982

: Reproduction is forbidden without the specific written :
: permission of ATARI, INC., Sunnyvale, CA 94086. No right :
: to reproduce this document, nor the subject matter thereof, :
: is granted unless by written agreement with, or written :
: permission from the Corporation.
:

C016555



ATARI®
PERSONAL COMPUTER SYSTEM
OPERATING SYSTEM
User's Manual

=====

NOTICE

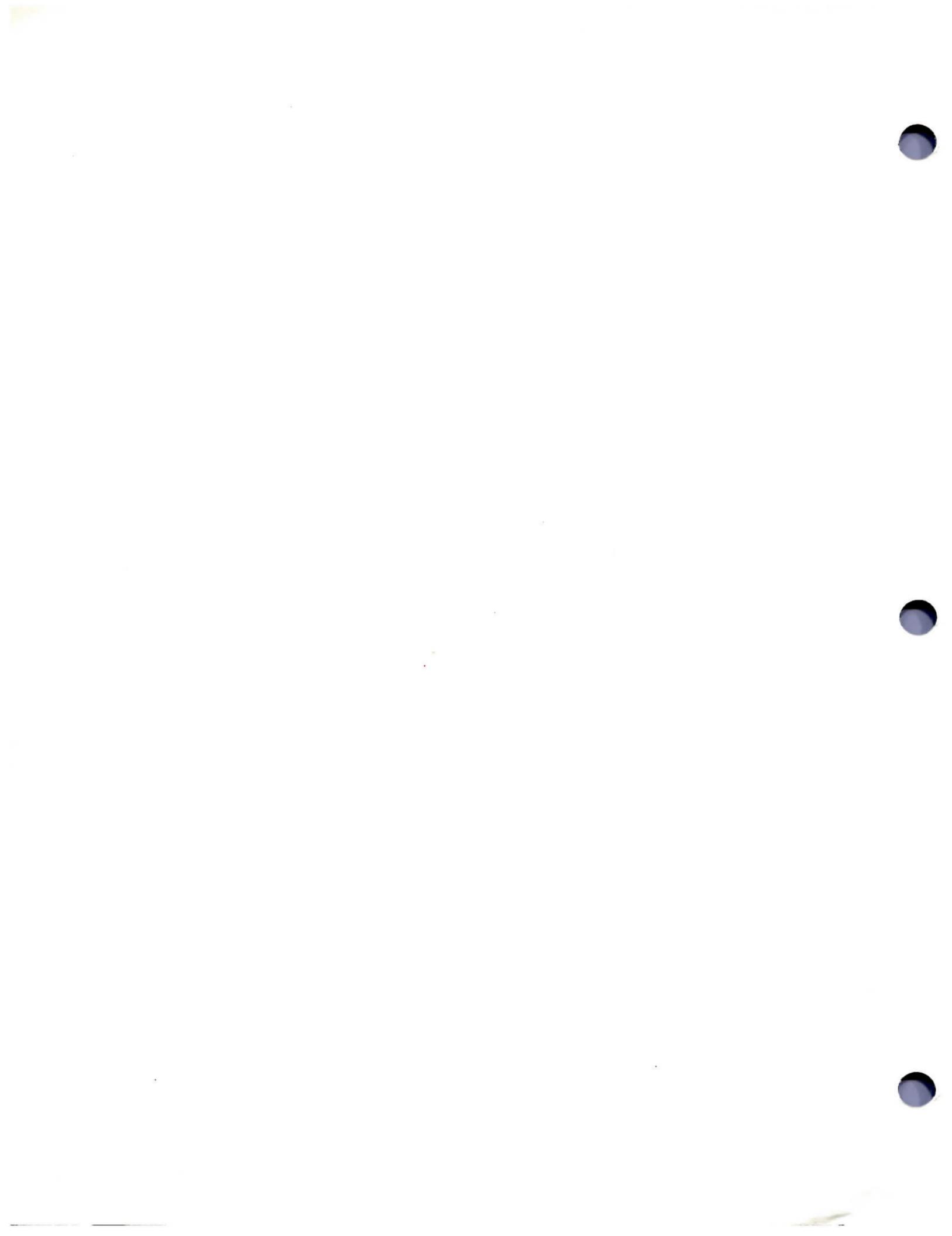
=====

TO ALL PERSONS RECEIVING THIS DOCUMENT

January 1982

: Reproduction is forbidden without the specific written :
: permission of ATARI, INC., Sunnyvale, CA 94086. No right :
: to reproduce this document, nor the subject matter thereof, :
: is granted unless by written agreement with, or written :
: permission from the Corporation.
:

C016555



REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

COPYRIGHT 1980 ATARI, INC.

Every effort has been made to ensure that this manual is an accurate document. However, due to the ongoing improvement and update of the computer software and hardware, ATARI, INC. cannot guarantee the accuracy of printed material after the date of publication, nor can ATARI, INC. accept responsibility for errors or omissions.



ATARI Personal Computer System
OPERATING SYSTEM USER'S MANUAL
TABLE OF CONTENTS

1. Introduction

Purpose of manual - (9)

General description of ATARI Personal Computer System - (9-11)
Notations used in this manual - (12-14)

2. Operating System functional organization

I/O subsystem - (15)

Interrupt processing - (15)

Initialization - (15)

Power up
RESET

Floating point arithmetic package - (16)

3. Configurations

Program environments

Blackboard mode
Cartridge
Disk boot
Cassette boot

RAM expansion

Peripheral devices

Game controllers
Cassette
Serial bus devices

4. System memory utilization

RAM region

Page 0
Page 1
OS database
User workspace
Boot region
Screen display list and data
Free memory region

Cartridges A and B
Mapped I/O
Resident OS and floating point package ROM
Central database description
Memory dynamics

System initialization process
Changing screen modes

5. I/O subsystem

Overview
Central I/O Utility (CIO)

CIO design philosophy
CIO calling mechanism
CIO functions
Device/filename specification
I/O example

Device specific information

Keyboard handler (K:)
Display handler (S:)
Screen Editor (E:)
Cassette handler (C:)
Printer handler (P:)
Disk File Manager (D:)
RS-232-C

Non-CIO I/O

Resident device handler vectors
Resident Disk handler
Educational System format cassettes
Serial bus I/O (SIO)

Device characteristics

Keyboard
Display
Cassette
Printer

Diskette
RS-232-C

6. Interrupt processing

Overview
Chip reset
Non-maskable interrupts (NMI)

Stage 1 VBLANK process
Stage 2 VBLANK process

Maskable interrupts (IRQ)

Interrupt initialization

System timers

Usage notes

POKEY interrupt mask
Setting interrupt and timer vectors
Stack content at interrupt vector points
Miscellaneous considerations

Flowcharts

7. System initialization

Overview
Power-up initialization (coldstart)
RESET initialization (warmstart)

8. Floating point arithmetic package

Description
Functions/calling sequences
Resource utilization
Implementation details

9. Adding new device handlers/peripherals

Device Table
CIO/Handler interface

Calling mechanism
Handler initialization
Functions supported
Error handling
Resource allocation

Handler/SIO interface

Calling mechanism
Functions supported
Error handling

Serial I/O bus characteristics and protocol

Hardware/electrical characteristics
Bus commands
Bus timing

Handler environment

Bootable handler
Cartridge resident handler

Flowcharts

10. Program environment and initialization

Cartridge

Cartridge without disk booted support package
Cartridge with disk booted support package

Disk booted software

Disk boot file format
Disk boot process
Sample disk bootable program listing
Program and procedure to create disk boot files

Cassette booted software

Cassette boot file format
Cassette boot process
Sample cassette bootable program listing
Program and procedure to create cassette boot files

11. Advanced techniques and application notes

Sound generation

 Capabilities
 Conflicts with OS

Screen graphics

 Hardware capabilities
 OS capabilities
 Cursor control
 Color control
 Alternate character sets

Players/missiles

 Hardware capabilities
 Conflicts with OS

Reading game controllers

 Keyboard controller sensing
 Front panel connectors as I/O ports



Appendices

- A -- CIO COMMAND BYTE values
- B -- CIO STATUS BYTE values
- C -- SIO STATUS BYTE values
- D -- ATASCII codes
- E -- Display codes (ATASCII)
- F -- Keyboard codes (ATASCII)
- G -- Printer codes (ATASCII)
- H -- Screen mode characteristics
- I -- Serial bus I.D. & command summary
- J -- ROM vectors
- K -- OS Database Variable Functional Descriptions
- L -- Equate file



1. Introduction

Purpose of manual

This manual provides a description of the Resident Operating System for the ATARI Personal Computer System, for use by persons concerned with the internal behavior of the systems. This manual discusses:

- o The functions provided by the system and how to use them
- o The organization of the various subsystems
- o The characteristics of the ATARI peripheral devices which may be attached
- o Standard techniques for going beyond the basic OS capabilities
- o The general nature of the hardware involved

This manual assumes that the reader is already familiar with programming concepts and jargon, assembly language programming in general and with the Synertek 6502 in particular, and has some degree of familiarity with digital hardware. The primary intent is to provide an experienced programmer with sufficient information so that he or she can effectively utilize the resources provided by the OS without having to resort to OS listings or trial and error techniques. A secondary goal is to provide supporting information for those individuals who do have to work with the OS listings.

This manual does not attempt to describe the hardware being used to provide the OS capabilities in any comprehensive fashion. Therefore, the person wanting to go beyond the capabilities described here are advised to examine the ATARI Personal Computer System HARDWARE MANUAL. This applies mostly to persons involved in the design of game cartridges, where display requirements, system timing and/or memory requirements preclude usage of the OS for one or more functions.

General description of the ATARI Personal Computer System

The ATARI 400TM and ATARI 800TM Personal Computer Systems are virtually identical from the standpoint of the operating system. In fact, the OS is identical in both models. The primary differences between the ATARI 400 and ATARI 800 Personal Computer Systems are:

- o Physical packaging
- o The ATARI 400 console has one cartridge slot, the ATARI 800 console has two slots
- o The ATARI 400 Personal Computer System can be expanded from 8k to 16K RAM maximum, the ATARI 800 Personal Computer System can be expanded to 48K RAM maximum

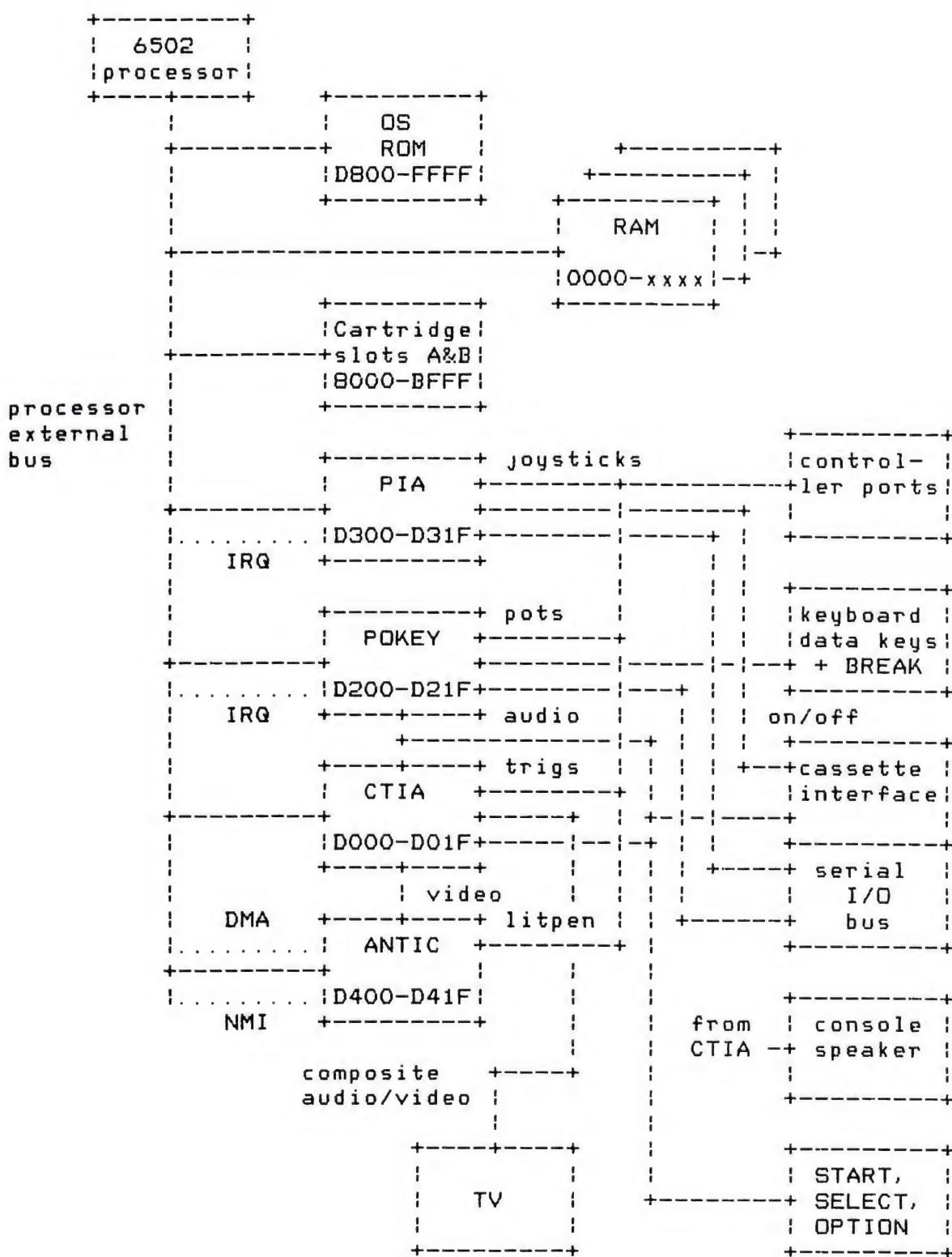
The hardware contains circuitry to: produce both character and point graphics for B&W or color television, produce four independent audio channels (frequency controlled) which use the television sound system, provide one bi-level audio output in the base unit, interface up to

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

4 joysticks, 8 paddle controllers/4 driving controllers, light pen, interface to a serial I/O bus for expansion, and has a built in keyboard. A simplified block diagram of the hardware is shown on the next page.

See the ATARI Personal Computer System HARDWARE MANUAL (part number CO16555) for supporting documentation.

ATARI Personal Computer System Block Diagram



Notations used in this manual

Several special notations are used throughout this manual in order to concisely present certain types of information such as hexadecimal numbers, memory addresses and data syntax. These notations are explained in the paragraphs that follow.

MEMORY ADDRESSES

All references to computer memory (and mapped I/O) locations will be in hexadecimal notation; sometimes the addresses will be contained in square brackets, such as 'D20F]', and sometimes not, such as 'D20F'

HEXADECIMAL NUMBERS

All two digit numbers preceded by a dollar sign ('\$') are to be read as hexadecimal numbers. Where not so prefixed, or specified otherwise by supporting text, a number that is not a memory address is expressed in decimal.

KILOBYTES OF MEMORY

Memory sizes are frequently expressed in units of kilobytes, such as 32K, where a kilobyte is 1024 bytes of memory.

PASCAL AS AN ALGORITHM SPECIFICATION LANGUAGE

In the few places where an algorithm is specified in detail, the Pascal language (procedure block only) is used as the specification language. Pascal syntax is similar to that of any number of other block structured languages, and the user should have no difficulty following the code presented.

MEMORY LAYOUTS

Whenever pictures of bytes or tables are presented, figures similar to the one shown below are used:

```
7 6 5 4 3 2 1 0
+---+---+---+---+
|           | --- this is a single byte.
+---+---+---+---+
|           |
+           + --- this is a word (2 bytes).
|           |
+---+---+---+---+
|           |
=           = --- this is a block of memory
|           | of unspecified length.
+---+---+---+---+
```

Where bit-7 is the most significant bit (msb) of the byte, and bit-0 is the least significant bit (lsb).

In table figures, memory addresses always increase toward the bottom of the figure.

BACKUS-NAUR FORM (BNF)

A modified version of BNF is used to express some syntactic forms, where the following meta-linguistic symbols are used:

::= is the substitution (assignment) operator.

< > bracket a metasyntactic variable.

| separates alternative substitutions.

[] bracket an optional construct.

anything else is a syntactic literal constant, which stands for itself.

For example:

```
<device specification> ::= <device name>[<device number>];
<device name> ::= C|D|E|I|K|P|R|S
<device number> ::= 1|2|3|4
```

The above statements specify that something called a "device specification" consists of a mandatory "device name" followed by an optional "device number" followed by the character ':'. The "device name", in turn, must be one and only one of the characters shown as alternatives; while the "device number" (if it is present) must be a digit 1 through 4.

OS EQUATE FILE NAMES

Operating system ROM and RAM vector names, RAM database variable names and hardware register names are all referred to herein by the names

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, C016555

assigned in the OS program equate list; in most cases, when one of these names is used, the memory address is provided also, such as 'BOOTAD [0242]'.

2. Operating System functional organization

This section describes the various subsystems of the resident OS in general terms.

I/O subsystem

The I/O subsystem provides a high-level interface between the programs and the hardware. Most functions are device independent, such as the reading and writing of character data; yet provisions have been made for device dependent functions as well. All peripheral devices capable of dealing with character data have individual symbolic names (such as K, D, P, etc.) and may be accessed using a Central I/O (CIO) routine.

Controllers such as joysticks, paddle controllers and the light pen, which do not deal with character data, are accessed via a RAM data base which is periodically updated to show the states of these devices.

Interrupt processing

All hardware interrupts are handled in a common and consistent manner by the Interrupt subsystem. By default, all interrupts are fielded by the OS, but at the discretion of the user, individual interrupts (or groups of interrupts) may be fielded by the application program.

Initialization

There are two levels of initialization provided by the system: power up and [RESET]. Power up initialization is performed each time the system power is turned on, and [RESET] initialization is performed each time the [RESET] key is pressed.

Power up

Whenever system power is turned on, the OS examines and notes the configuration of the unit; the following items are among those things performed at power up:

- o Determine the highest RAM address.
- o Clear all of RAM to zeroes.
- o Establish all RAM interrupt vectors.
- o Format the Device Table.
- o Initialize the cartridge(s).
- o Setup the screen for 24 x 40 text mode.
- o Boot the cassette if directed.
- o Check cartridge slot(s) for disk boot instructions.
- o Boot the disk if directed to do so and a disk is attached.
- o Transfer control to the cartridge, disk booted program, cassette booted program, or blackboard program.

[RESET]

Whenever the [RESET] key is pressed, the OS performs some, but not all, of the functions performed at power up; the following items are among those things performed as a result of pressing the [RESET] key:

- o Clear the OS portion of RAM.
- o Re-establish all RAM interrupt vectors.
- o Format the Device Table.
- o Initialize the cartridge(s).
- o Setup the screen for 24 x 40 text mode.
- o Transfer control to the cartridge, disk booted program, cassette booted program, or blackboard program.

Floating point arithmetic package

Contained within the OS ROM is a floating point (FP) package which is not used by the other parts of the OS itself, but is available to non-resident programs such as BASIC, Calculator, Pascal, etc. The floating point numbers are stored as 10 BCD digits of mantissa plus a 1 byte exponent. The following routines are among those found in the package:

- o ASCII to FP and FP to ASCII conversion.
- o Integer to FP and FP to integer conversion.
- o FP add, subtract, multiply and divide.
- o FP log, exp and polynomial evaluation.
- o FP number clear, load, store and move.

3. Configurations

The ATARI 400 and ATARI 800 Personal Computer Systems support a wide variety of configurations, each with a unique operating environment: cartridge(s) may or may not be inserted, memory may be optionally added to the ATARI 800 computer console in 8K or 16K increments, and many different peripheral devices may be attached to the serial I/O bus. The OS accounts for all of these variables without requiring a change in the resident OS itself. As explained in Section 2, the machine configuration is checked when power is first turned on and then is not checked again. A general discussion of some of the valid configurations follows.

Program environments

The OS allows one of four program types to be in control at any point in time: the OS blackboard (ATARI Memo Pad) program, a cartridge resident program, a disk booted program or a cassette booted program. Which one of these is in control is based upon information in the cartridge(s), whether or not a disk is attached and operator keyboard inputs; the exact algorithms are discussed in detail in section 7.

Blackboard mode

When in blackboard mode, the screen is established as a 24 x 40 text screen. Anything entered from the keyboard goes to the screen without being examined; although all of the screen editing functions are supported. Blackboard mode is the lowest priority environment; one only goes there if there is no other reasonable environment for the OS to enter or if the operator requests a higher priority environment to enter the blackboard mode (for example, BYE in BASIC). If it was entered from a higher priority environment, the blackboard mode may be exited by pressing the [RESET] key.

Cartridge

When a cartridge is inserted, it normally provides the main control after initialization is complete; for example: BASIC, Super Breakout®, BASKETBALL, COMPUTER CHESS, etc. all interface directly with the user in some way. Although it is possible for a cartridge to provide a supporting function for some other program environment, this has not yet been done. In some cartridges, particularly keyboard oriented ones, it possible to change environments by entering special commands such as "BYE" to go to blackboard mode or "DOS" to enter the Disk Utility. In many other cartridges, particularly games, it is not possible to change environments. Note that because of a hardware interlock it is impossible to remove or insert a cartridge with the power on; this means (among other things) that every cartridge change will completely reinitialize the entire system.

Disk boot

When the system powers up with a disk attached with disk bootable software, the disk may or may not be booted, depending upon conditions explained in section 7. The rest of this paragraph assumes that a disk boot did occur.

The disk booted software may take control as the Disk Utility does under certain conditions, or may provide a supporting function as the File Management System does; this environment is so flexible that it is difficult to generalize on its capabilities and restrictions. The only machine requirement (other than the disk drive) is that sufficient RAM be installed to support the program being booted.

Cassette boot

Everything that was said about the disk boot environment is also true about the cassette boot environment, although the cassette is limited as an I/O device due to its slowness, sequential access and single file at a time nature. Those limitations probably limit cassette booted software to "cartridge type" programs, 100 percent RAM resident and not involving random access nor much I/O involving permanent storage. Note that the cassette boot facility has no relation to the use of cassettes to store high level language programs (e.g. programs written in BASIC) nor to the use of cassettes to store data.

RAM expansion

Although RAM may be expanded non-contiguously by the user in the ATARI 800 Personal Computer System, the OS will only recognize RAM that is contiguous starting from location 0. Directions for installing the RAM modules are provided with the purchased modules. RAM may be added until it totals 48K; after 32K, additional RAM overlays first the right cartridge addresses (32K to 40K) and then the left cartridge addresses (40K to 48K). Note that in cases of conflict, the inserted cartridge has higher priority and disables the conflicting RAM in 8K increments. See section 4 for a detailed discussion of system memory.

As a result of power up the OS will generate two pointers that define the lowest available RAM location and the highest available RAM location. The OS and disk/cassette booted software will determine the location of the lowest available RAM, while the number of RAM modules and the current screen mode will determine the highest available RAM.

Peripheral devices

Peripheral devices of several types may be added to the system using standard cables to either the serial bus or the connectors at the front of the computer console. The most common types deal with either

transmission of bytes of data (usually serial bus) or transmission of sense information (usually game controllers).

Game controllers

The standard game controllers (pots, joysticks, driving controllers, light pen, etc.) are sensed periodically (50 or 60 times per second) by the OS and the values read are stored in RAM. These controllers may be plugged in, pulled out, and rearranged at will by the user without affecting system operation; the system will always try to read all of these controllers. Other controllers, such as the keyboard controller, are not read by the OS and special instructions as described in section 11 are required to read them.

Cassette

The cassette is a special peripheral in that it uses the serial bus to send and receive data, but does not conform to the protocol of the other peripherals that use the serial bus. The cassette must also be the last device on the serial bus because it does not have a serial bus extender connector as the other peripherals do. The lack of a bus extender assures that there is never more than one cassette drive connected to any system. The system cannot sense the absence or presence of the cassette drive, so it may be connected and disconnected at will.

Serial bus devices

By serial bus devices we mean those that conform to the serial I/O bus protocol as defined in section 9; this does not include the cassette drive. Each serial bus device has two identical connectors: one a serial bus input, the other a serial bus extender. Either connector may be used for either purpose, and peripherals may be "daisy chained" simply by cabling them together in a sequential fashion. There are usually no restrictions on the cabling order, as each device has a unique identifier; where there are restrictions, they will be mentioned in section 5.

4. System memory utilization

Memory in the system is decoded in the full 64K range of the 6502 microcomputer and there are no provisions for additional mapping to extend memory. Memory is divided into four basic regions (with some overlap possible): RAM, cartridge area, I/O region and the resident OS ROM. The regions and their address boundaries are listed below (all addresses are in hexadecimal):

0000-1FFF = RAM (minimum required for operation)
2000-7FFF = RAM expansion area
8000-9FFF = Cartridge B, Cartridge B (half of 16K size) or RAM
A000-BFFF = Cartridge A or RAM
C000-CFFF = unused
D000-D7FF = Hardware I/O decodes
D800-DFFF = Floating point package (OS)
E000=FFFF = Resident Operating System ROM

This section will break these regions into even smaller functional divisions and provide detailed explanations of their usage.

RAM region

The RAM region is shared between the OS and the program in control and can be further subdivided into the following sub-regions for discussion purposes:

Page 0 = 6502 page zero address mode region.
Page 1 = 6502 stack region.
Pages 2-4 = OS database & user workspace.
Pages 5-6 = User program workspace.
Pages 7-XX = Bootable software area/free RAM*.
Pages XX-top of RAM = screen display list and data*.

* Note that XX is a function of the screen graphics mode and the amount of RAM installed.

The paragraphs that follow indicate the OS usage and recommended user program usage of these RAM sub-regions.

Page 0

Because of the architecture of the 6502 microcomputer instruction set and addressing modes, page 0 has special significance; references to addresses in that page (0000 to 00FF) are faster, require fewer instruction bytes and provide the only mechanism for hardware indirect addressing. Therefore page 0 is a resource that has to be utilized sparingly so that all possible users may have portion of it. The OS permanently takes the lower half of page 0 (0000 to 007F) and this portion may never be used by any outer environment unless the OS is completely disabled and all interrupts to the OS are eliminated.

The upper half of page 0 (0080 to 0OFF) is available to outer environments with the following restriction: the floating point package, if used, requires 00D4 through 0OFF.

Page 1

Page 1 is the 6502 hardware stack region; JSR instructions, PHA instructions and interrupts all cause data bytes to be written to page 1 and conversely RTS, PLA, and RTI instructions all cause data bytes to be read from page 1. The 256 byte stack is adequate for normal subroutine calls plus interrupt process nesting, so no restrictions have been made on page 1 usage. It is obvious that a stack of this size is totally inadequate for deeply recursive processes or for nested processes with large local environments to be saved. So, for sophisticated applications, software maintained stacks must be implemented.

The 6502 stack pointer is initialized at power up or [RESET] to point to location 01FF, the stack then pushes downward toward 0100. The stack will wrap around from 0100 to 01FF if a stack overflow condition occurs, due to the nature of the 6502's 8-bit stack pointer register.

OS database

Locations 0200 through 047F are allocated by the OS for working variables, tables and data buffers. Portions of this region may be used only after it is determined that nonconflict with the OS is guaranteed. For example, the printer and cassette buffers could be used if I/O operations to these devices are impossible within the controlling environment. The amount of work involved in determining nonconflict seems to be completely out of line with the benefits to be gained (except for a few trivial cases) and it is recommended that pages 2 through 7 not be used except by the OS.

User workspace — PAGES 5-6

4

Locations 0480 through 06FF are dedicated for outer environment use except when the floating point package is used, in which case it uses locations 057E through 05FF.

Boot region

Page 7 is the start of the "boot region". When software is booted from either the disk or the cassette, it may start at the lowest free memory address (which is 0700) and proceed upward (although it may also start at any address above 0700 and below the screen display list). The top of this region defines the start of the "free memory" region. When the boot process is complete, a pointer in the data base contains the address of the next available location above the software

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

just booted. When no software has been booted, this pointer contains the value 0700.

Screen display list and data

When the OS is handling the screen display, the display list which defines the screen characteristics and the current data which is contained on the screen are placed at the high address end of RAM. The bottom of this region defines the end of the "free memory" region and its location is a function of the screen mode currently in effect. A pointer in the data base contains the address of the last available location below the screen region.

Free memory region

The free memory region is all that RAM between the end of the boot region and the start of the screen region. The outer level application is responsible for managing the free memory region. See section 4 for more details.

Cartridges A and B

There are two 8K regions reserved for plug-in cartridges. Cartridge B, which is the right hand cartridge slot found only in the ATARI 800 Personal Computer System, has been allocated memory addresses 8000 through 9FFF; while cartridge A, which is the left-hand cartridge slot in the ATARI 800 computer console, and is the only slot in the ATARI 400 computer console, has been allocated memory addresses A000 through BFFF and optionally 8000 through BFFF, for 16K cartridges. If a RAM module is plugged into the last slot such as to overlay any of these addresses, the RAM takes precedence as long as a cartridge is not inserted. However, if a cartridge is inserted, it will disable the entire conflicting RAM module in the last slot in 8K increments.

Mapped I/O

The 6502 performs input/output operations by addressing the external support chips as memory; some chip registers are read/write while others are read only or write only (the ATARI Personal Computer System HARDWARE MANUAL gives descriptions of all of the external registers). While the entire address space from D000 to D7FF has been allocated for I/O decoding, only the following sub-regions are used:

D000-D01F = CTIA
D200-D21F = POKEY
D300-D31F = PIA
D400-D41F = ANTIC

Resident OS and floating point package ROM

The region from D800 through FFFF always contains the OS and the floating point package. To allow for the possibility that another, but functionally compatible, OS may be generated in the future, care should be taken to avoid using any entry points that are not guaranteed not to move. The OS contains many vectored entry points at the end of the ROM and in RAM which will not move. The floating point package is not vectored, but all documented entry points will be fixed (this means do not use undocumented routines found by scanning the listing!). A list of the fixed ROM vectors and entry points may be found in Appendix J.

Central data base description

Discussion of organization of this section.

There are a large number of variables in the OS data base, most of which have some relevance to the user, either for control or debug purposes. This section provides detailed information for those variables which can be altered by the user in meaningful ways and to provide at least a narrative description of the remaining variables. One major problem, when dealing with this many variables, is how to present the information so that it is accessible to the user in the different contexts in which the user may work. This manual attempts to solve that problem by providing a multiple access scheme, in which several lookup tables are provided, all of which reference a common set of narratives that is itself ordered by function.

In order to provide a means of referencing the variable descriptions, the variable descriptions are each provided with a label consisting of a single letter followed by a number (e.g. A4, B13, etc.). A different letter is assigned for each major functional area being described, and the numbers are assigned sequentially within each functional area. This label just described will be referred to as a VID (variable identifier) throughout the remainder of this document. Those variables which are not considered to be of interest to any user are flagged with an asterisk ('*') after their names.

The database lookup tables provided are:

1. Functional grouping -- index to the function narrative and descriptions of variables, giving VID and variable name.

For more information, see Appendix K.

FUNCTIONAL INDEX TO DATABASE VARIABLE DESCRIPTIONS

A. Memory configuration

A1 MEMLO
A2 MEMTOP
A3 APPMHI
A4 RAMTOP*
A5 RAMSIZ

B. Text/graphics screen

Cursor control

B1 CRSINH
B2 ROWCRS, COLCRS
B3 OLDROW, OLDCOL
B4 TXTROW, TXTCOL

Screen margins

B5 LMARGN
B6 RMARGN

Color control

B7 PCOLR0 - PCOLR3
B8 COLOR0 - COLOR4

Text scrolling

B9 SCRFLG*

Attract mode

B10 ATRACT
B11 COLRSH*
B12 DRKMSK*

Tabbing

B13 TABMAP

Logical text lines

B14 LOGMAP*
B15 LOGCOL*

Split screen

B16 BOTSCR*

FILL/DRAW function

B17 FILDAT
B18 FILFLG*
B19 NEWROW*, NEWCOL*
B20 HOLD4*
B21 ROWINC*, COLINC*
B22 DELTAR*, DELTAC*
B23 COUNTR*

B24 ROWAC*, COLAC*
B25 ENDP†*

Displaying control characters

Escape (display following control char)
B26 ESCFLG*

Display control characters mode
B27 DSPFLG

Bit mapped graphics
B28 DMASK*
B29 SHFAMT*

Internal working variables

B30 HOLD1*
B31 HOLD2*
B32 HOLD3*
B33 TMPCHR*
B34 DSTAT*
B35 DINDEX*
B36 SAVMSC
B37 OLDCHR*
B38 OLDAADR*
B39 ADRESS*
B40 MLTTMP/OPNTMP/TOADR*
B41 SAVADR/FRMADR*
B42 BUFCNT*
B43 BUFSTR*
B44 SWPFLG*
B45 INSDAT*
B46 TMPROW*, TMPCOL*
B47 TMPLBT*
B48 SUBTMP*
B49 TINDEX*
B50 BITMSK*
B51 LINBUF*
B52 TXTMSC
B53 TXTOLD*

Internal character code conversion

B54 ATACHR
B55 CHAR*

C. Disk handler
C1 BUFADR*
C2 DSKTIM*

D. Cassette (part in SIO part in handler)

Baud rate determination

D1 CBAUDL*, CBAUDH*

D2 TIMFLG*

D3 TIMER1*, TIMER2*

D4 ADDCOR*

D5 TEMP1*

D6 TEMP3*

D7 SAVIO*

Cassette mode

D8 CASFLG*

Cassette buffer

D9 CASBUF*

D10 BLIM*

D11 BPTR*

Internal working variables

D12 FEOF*

D13 FTYPE*

D14 WMODE*

D15 FREQ*

E. Keyboard

Key reading and debouncing

E1 CH1*

E2 KEYDEL*

E3 CH

Special functions

Start/stop

E4 SSFLAG

[BREAK]

E5 BRKKEY

SHIFT/CONTROL lock

E6 SHFLOK

E7 HOLDCH*

Auto-repeat

E8 SRTIMR*

Inverse video

E9 INVFLG

Console switches (SELECT, START & OPTION)

F. Printer

Printer buffer

F1 PRNBUF*
F2 PBUFSZ*
F3 PBPNT*

Internal working variables

F4 PTEMP*
F5 PTIMOT*

G. Central I/O routine (CIO)

User call parameters

G1 IOCB
G2 ICHID
G3 ICDNO
G4 ICCOM
G5 ICSTA
G6 ICBAL, ICBAH
G7 ICPTL, ICPTH
G8 ICBLL, ICBLH
G9 ICAX1, ICAX2
G10 ICSPR

Device status
G11 DVSTAT

Device Table
G12 HATABS

CIO/handler interface parameters

G13 ZIOCB (IOCBAS)
G14 ICHIDZ
G15 ICDNOZ
G16 ICCOMZ
G17 ICSTAZ
G18 ICBALZ, ICBALH
G19 ICPTLZ, ICPTHZ
G20 ICBLLZ, ICBLHZ
G21 ICAX1Z, ICAX2Z
G22 ICSPRZ (ICIDNO, CIOCHR)

Internal working variables

G23 ICCOMT*
G24 ICIDNO*
G25 CIOCHR*

H. Serial I/O routine (SIO)

User call parameters

H1 DCB control block
H2 DDEVIC
H3 DUNIT
H4 DCOMND
H5 DSTATS
H6 DBUFLO, DBUFHI
H7 DTIMLO
H8 DBYTLO, DBYTHI
H9 DAUX1, DAUX2

Bus sound control
H10 SOUNDR

Serial bus control

Retry logic
H11 CRETRY*
H12 DRETRY*

Checksum
H13 CHKSUM*
H14 CHKSNT*
H15 NOCKSM*

Data buffering

General buffer control

H16 BUFRLO*, BUFRHI*
H17 BFENLO*, BFENHI*

Command frame output buffer
H18 CDEVIC*
H19 CCOMND*
H20 CAUX1*, CAUX2*

Receive/transmit data buffering
H21 BUFRFL*
H22 RECVDN*
H23 TEMP*
H24 XMTDON*

SIO timeout
H25 TIMFLG*
H26 CDTMV1*
H27 CDTMA1*

Internal working variables
H28 STACKP*
H29 TSTAT*

H30 ERRFLG*
H31 STATUS*
H32 SSKCTL*

J. ATARI controllers

Joysticks
J1 STICK0 - STICK3
J2 STRIGO - STRIG3

Paddles
J3 PADDL0 - PADDL7
J4 PTRIGO - PTRIG7

Light pen
J5 LPENH
J6 LPENV
J7 STICK0

Driving controllers
J8 STICK0 - STICK3
J9 STRIGO - STRIG3

K. Disk file manager

K1 FMSZPG*
K2 ZBUFP*
K3 ZDRVA*
K4 ZSBA*
K5 ERRNO*

L. Disk utilities (DOS)

L1 DSKUTL*

M. Floating point package

M1 FRO
M2 FRE*
M3 FR1
M4 FR2*
M5 FRX*
M6 EEXP*
M7 NSIGN*
M8 ESIGN*
M9 FCHRFLG*
M10 DIGRT*
M11 CIX
M12 INBUFF
M13 ZTEMP1*
M14 ZTEMP4*
M15 ZTEMP3*
M16 FLPTR

M17 FPTR2*
M18 LBPR1*
M19 LBPR2*
M20 LBUFF
M21 PLYARG*
M22 FPSCR/FSCR*
M23 FPSCR1/FSCR1*
M24 DEGFLG/RADFLG*

N. Power up & [S RESET]

RAM sizing
N1 RAMLO*, TRAMSZ*
N2 TSTDAT*

Disk/cassette boot

N3 DOSINI
N4 CKEY*
N5 CASSBT*
N6 CASINI
N7 BOOT?*
N8 DFLAGS*
N9 DBSECT*
N10 BOOTAD*

Environmental control

N11 COLDST*
N12 DOSVEC

[S RESET]

N13 WARMST

P. Interrupts

P1 CRITIC
P2 POKMSK

System timers

Real-time clock
P3 RTCLOK

System timer 1
P4 CDTMV1
P5 CDTMA1

System timer 2
P6 CDTMV2
P7 CDTMA2

System timers 3-5
P8 CDTMV3, CDTMV4, CDTMV5
P9 CDTMF3, CDTMF4, CDTMF5

RAM interrupt vectors

NMI interrupt vectors

P10 VDSLST
P11 VVBLKI
P12 VVBLKD

IRQ interrupt vectors

P13 VIMIRQ
P14 VPRCED
P15 VINTER
P16 VBREAK
P17 VKEYBD
P18 VSERIN
P19 VSEROR
P20 VSEROC
P21 VTIMR1, VTIMR2, VTIMR4

Hardware register updates

P22 SDMCTL*
P23 SDLSTL*, SDLSTH*
P24 GPRIOR*
P25 CHACT*
P26 CHBAS
P27 PCOLRx, COLORx

Internal working variable

P28 INTEMP*

R. User areas

R1 (unlabeled)
R2 USAREA

S. Unused (spare) bytes

S1 HOLDS
S2 CSTAT
S3 DUNUSE
S4 TEMP2
S5 TMPX1
S6 DSKFMS
S7-S15 (unlabeled)

Memory dynamics

The free memory region is the area between the end of the boot region and the start of the screen region, and as such, its limits are variable. The bottom of the free region is defined by the content of the variable MEMLO [02E7], and the top of the region is defined by the content of the variable MEMTOP [02E5]. The conditions which cause the setup or alteration of these variables are now discussed.

System initialization process

When the system is powered-up, the extent of the lowest block of contiguous RAM is determined and the limits are saved. The Screen Editor is then opened, thus setting a new (and lower) value in MEMTOP. Then, as discussed in section 7, disk or cassette booted software might be brought into memory, which would probably set a new (and higher) value in MEMLO. When the application program finally gets control, MEMLO and MEMTOP will define the maximum amount of free memory available at that time; however, that amount may later decrease further, as described in the next paragraph.

Changing screen modes

The user may, at any time, command the Display handler to change screen modes. In most cases this will involve a change in the memory required for the display list and display data, and hence, will change the value of MEMTOP. Appendix H indicates the amount of memory required for each of the screen modes.

In order to allow the user to protect the portion of free memory space that he is using from being overwritten as a result of a screen mode change, the variable APPMHI [000E] is interpreted by the Display handler as containing the address below which MEMTOP may not extend. If, as a result of a screen mode change, the Display handler determines that the screen memory would extend below APPMHI, then the screen is setup for mode 0, MEMTOP is updated and an error status is returned to the user; otherwise the desired mode change is effected and MEMTOP is updated.

5 I/O subsystem

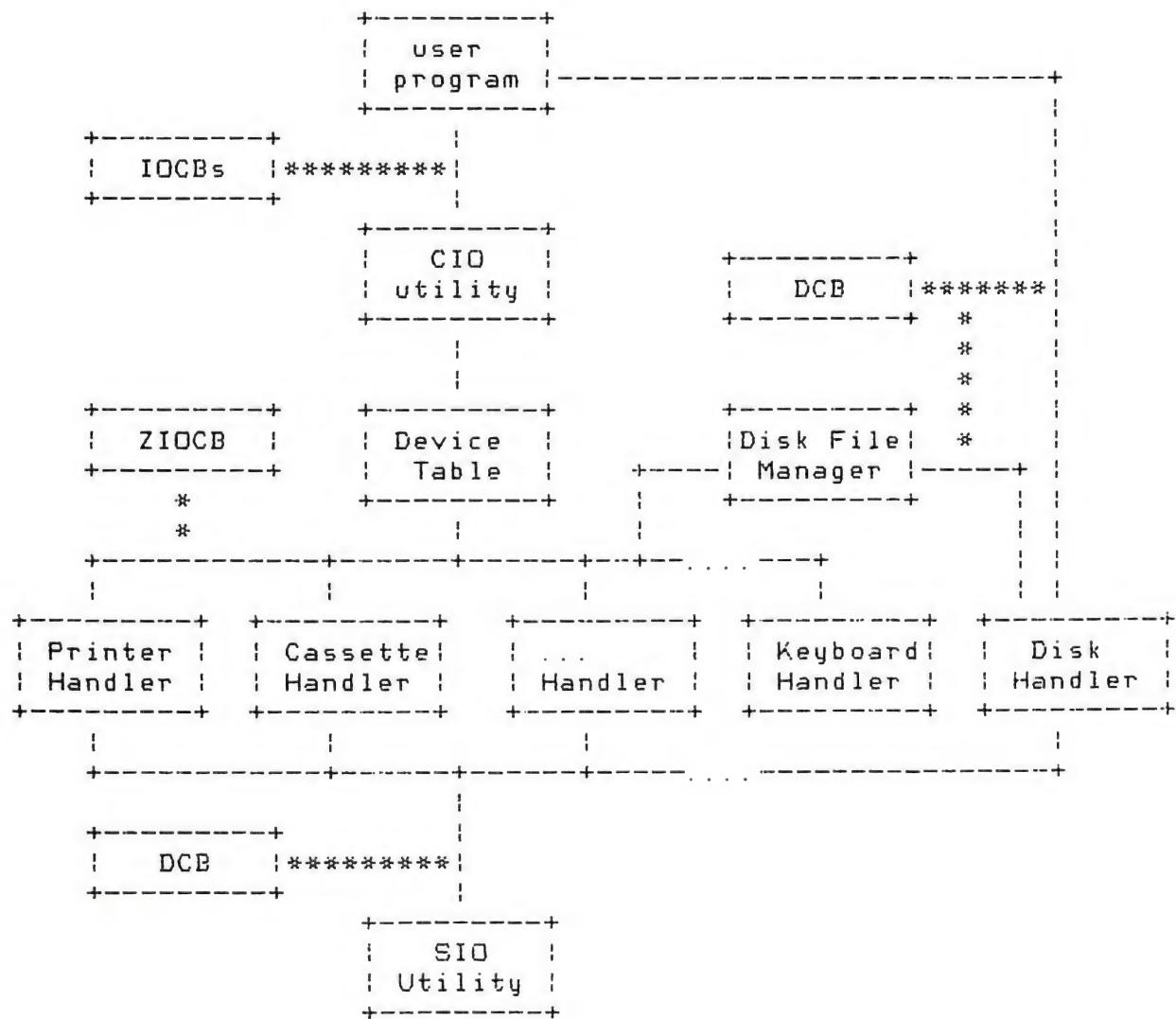
Introduction

This section discusses the I/O subsystem of the operating system. The I/O subsystem is a collection of routines which allows the user to access peripheral and local devices at a number of different levels, all higher than that of accessing the device hardware registers directly. The routine of interest to most users is CIO (Central I/O utility), which provides the highest level, device independent access to devices. The next level down would be communication with the device handlers, followed by use of the SIO (Serial I/O bus utility) routine, which is the bottom level general I/O routine in the OS. Any lower level access to a device would involve the direct reading and writing of the hardware registers associated with the device.

The basic unit of input/output is the data byte, which can contain either "binary" (non-text) information or encoded text information. The text encoding scheme supported by the OS is called ATASCII, the name of which is derived from the words 'ATARI ASCII'. Most ATASCII codes are the same as ASCII, with the primary deviations being the control codes. Appendix D shows the ATASCII character set, and Appendices E, F and G show device specific implementations for the display, keyboard and printer.

The structure of the I/O subsystem is shown on the following page.

I/O SUBSYSTEM FLOW DIAGRAM



Where: ---- shows a control path.

**** shows the data structure required for a path.

Note the following:

1. The Keyboard/Display/Screen Editor handlers don't use SIO.
2. The Disk handler is not callable directly from CIO.
3. The DCB is shown twice in the diagram.

Central I/O Utility (CIO)

The Central I/O Utility (CIO) provides the user with a single interface with which to access all of the system peripheral devices, in a device independent manner. The minimum unit of data transfer is the data byte, with multiple byte transfers also supported. All I/O

operations are performed on a "return to user when complete" basis; there is no way to initiate concurrent "overlapped" I/O processes.

I/O is organized by "files", where a file is a sequential collection of data bytes. A file may or may not contain textual data and it may or may not be organized by "records", where a record is a contiguous group of bytes terminated by an EOL (End of Line) character. Some files are synonymous with a device (as with the printer and the Screen Editor), while other devices may contain a multiplicity of files, each with a unique name (as with the floppy disk).

CIO will allow the user to access up to eight independent device/files at one time; there being that many I/O Control Blocks (IOCBs) in the system. Each of the IOCBs may be assigned to control any desired device/file, as there are no preferred assignments, except that IOCB #0 is assigned to the Screen Editor at power up and [S/RESET].

In order to access a peripheral, the user must first setup an IOCB for the OPEN command, which supplies the system name for the device to be accessed (e.g. 'K:' for the keyboard, 'P:' for the printer, 'D:STARS' for a disk file named 'STARS', etc.). The user then calls CIO, telling it which IOCB to use to find the OPEN information. CIO attempts to find the specified device/file and returns a status byte indicating the success of the search. If the specified device/file can be found by CIO, then CIO stores control information in the IOCB and that IOCB is now used for as long as that file is OPEN.

Once a file is OPEN, it can then be accessed using data read or data write types of commands; in general, reading may proceed until there is no more data to read (End of File) and writing may proceed until there is no more medium to store data on (End of Medium), although neither reading nor writing need proceed to that point. The reading and writing of data generally occurs into and out of user supplied data buffers (although a special case allowing single byte transfers using the 6502 A register is provided).

When there are no more accesses to be performed on an OPEN device/file, the CLOSE operation is performed by the user. This accomplishes two functions: 1) it terminates and makes permanent an output file (essential for disk and cassette) and 2) it releases that IOCB to be used for another I/O operation.

CIO Design Philosophy

The CIO utility was designed specifically to meet the following design criteria.

The transfer of data is device independent.

Byte-at-a-time, multiple byte and record aligned accesses are supported.

Multiple device/files can be accessed concurrently.

Error handling is largely device independent.

New device handlers may be added without altering the system ROM.

DEVICE INDEPENDENCE

CIO provides device independence by having a single entry point for all devices (and for all operations) and by having a device independent calling sequence. Once a device/file is OPENed, data transfers occur with no regard to the actual device involved. Uniform rules for handling byte and record oriented data transfers allow the actual device storage block sizes to be transparent to the user.

DATA ACCESS METHODS

Two file access methods are supported by CIO: byte aligned and record aligned.

Byte aligned accesses allow the user to treat the device/file as a sequential byte stream; any number of bytes may be read or written and the following operation will continue where the prior one left off. Records are of no consequence in this mode, and reads or writes may encompass multiple records if desired.

Record aligned accesses allow the user to deal with the data stream at a higher level, that of the data record or "line of text". Each and every write operation creates a single record (by definition), and each read operation assures that the following read operation will start at the beginning of a record. Record aligned accesses may not deal with portions of more than one record at a time. Record aligned accesses are useful only with text data or with binary data guaranteed not to contain the EOL character (\$9B) as data.

Note that any file may be accessed using the byte aligned access method, regardless of how the file was created. But not all files may be successfully read using record aligned accesses; the file must contain EOL characters at the end of each record and at no other place.

MULTIPLE DEVICE/FILE CONCURRENCY

Up to eight device/files may be accessed concurrently using CIO, each operating independently of the others.

UNIFIED ERROR HANDLING

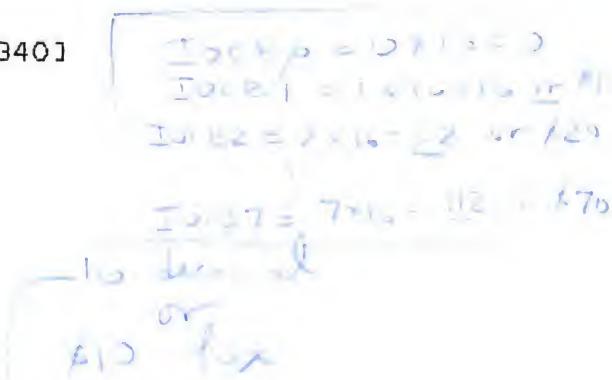
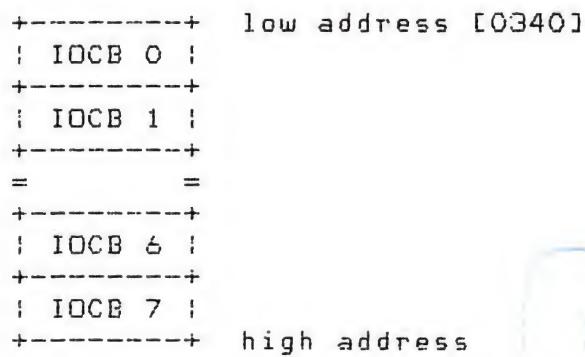
All error detection and recovery occurs within the CIO subsystem and the status information that reaches the user is in the form of a status byte for each device/file. As much as possible, error codes are device independent (see Appendix B). — *PAGE 165*

DEVICE EXPANSION

Devices are known by one character names such as 'K' or 'P', and a number of device handlers are part of the resident system ROM. However, additional device handlers may be added to the system using the RAM resident Device Table; this is normally done at power up time as with the disk boot process, but may be done at any point in time.

CIO calling mechanism

The primary element in performing I/O using CIO is an Input/Output Control Block (IOCB). There are eight IOCBs in the system, arranged linearly in RAM as shown below.



One IOCB is required for each OPEN device/file and any IOCB may be used to control any device/file; although IOCB 0 is normally assigned to the Screen Editor (E:). A typical I/O operation is performed by having the user: 1) insert appropriate parameters into an IOCB of his choosing, 2) put the IOCB number times 16 into the 6502 X register and 3) JSR to the CIO entry point CIOV [E456]. CIO will return to the user when the operation is complete or if an error was encountered; the status of the operation will be in the IOCB used as well as in the 6502 Y register; in addition, the 6502 conditions codes will reflect the value in the Y register.. In some cases a data byte will be in the 6502 A register. The X register will remain unchanged for all operations and conditions. An example is shown below:

```
IOCB2X = $20 ; INDEX FOR IOCB #2.  
  
LDX    #IOCB2X  
JSR    CIOV  
CPY    #0; (optional)  
BMI    ERROR
```

Each IOCB is sixteen bytes long, in which some bytes are user alterable and some are for use by CIO and/or the device handlers. Each of the IOCB bytes will now be described, and the system equate file name and memory address for each will be given.

HANDLER I.D. -- ICHID [0340]

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

The handler I.D. is an index into the system Device Table (see section 9) and is not user alterable. This byte is set by CIO as the result of an OPEN command and is left unchanged until the device/file is CLOSED, at which time CIO will set the byte to \$FF.

DEVICE NUMBER -- ICDNO [0341]

The device number is provided by CIO as the result of an OPEN command and is not user alterable. This byte is used to distinguish between multiple devices of the same type, such as 'D1:' and 'D2:'.

COMMAND BYTE -- ICCMD [0342]

The command byte is set by the user and tells CIO which of its repertoire of commands is to be performed. The commands and their command byte values will be detailed in section 5.2.3. This byte is not altered by CIO.

STATUS -- ICSTA [0343]

The status byte is used by CIO to convey operation status to the user; it is updated as a result of each and every CIO call. The most significant (sign) bit is a one for error conditions and zero for non-error conditions, and the remaining bits represent an error number. See Appendix B for a list of status codes.

BUFFER ADDRESS -- ICBAL [0344] & ICBAH [0345]

This two byte pointer is set by the user and is not altered by CIO. The pointer contains the address of the beginning (low address) of a buffer which is used to: 1) contain data for read and write operations and 2) contain the device/filename specification for the OPEN command. The pointer may be altered at any time by the user.

PUT ADDRESS -- ICPTL [0346] & ICPTH [0347]

This two byte pointer to the handler's PUT CHARACTER entry point (- 1) is set by CIO at OPEN time; this was provided as an accommodation to the people writing the BASIC cartridge and has no legitimate use in the system. This variable is set to point to CIO's "IOCB not OPEN" routine on CLOSE, power up and [S/RESET].

BUFFER LENGTH/BYTE COUNT -- ICBLL [0348] & ICBLH [0349]

This two byte count is set by the user to indicate the size of the data buffer pointed to by ICBAL and ICBAH for read and write operations; it is not required for OPEN. After each read or write operation, CIO will set this parameter to the number of bytes actually transferred into or out of the data buffer. For record aligned accessed, the record length may well be less than the buffer length. Also an end of file condition or an error may cause the byte count to be less than the buffer length.

AUXILLIARY INFORMATION -- ICAX1 [034A] & ICAX2 [034B]

These two bytes are set by the user and contain information which is used by the OPEN command process and/or is device dependent.

For OPEN, two bits of ICAX1 are always used to specify the OPEN direction as shown below, where R is set to 1 for input (read) enable and W is set to 1 for output (write) enable.

7	3	2 0
+---+---+---+---+---+		
		W R
+---+---+---+---+---+		

ICAX1 is not altered by CIO and should not be altered by the user once the device/file is OPEN.

The remaining bits of ICAX1 and all of ICAX2 contain only device dependent data and are explained in section 5:Device specific information.

REMAINING BYTES (ICAX3-ICAX6)

The four remaining bytes are reserved for use by the handler processing the I/O command for CIO. There is no fixed use for these bytes and they are not user alterable except as specified by the particular device descriptions in section 5.3. These bytes will be referred to as ICAX3, ICAX4, ICAX5 and ICAX6, although there are no equates for those names in the OS equate file.

CIO functions

There are eight basic functions that are supported by all of the system handlers, subject to restrictions based upon the direction of data transfer (e.g. one cannot read data from the printer). The basic functions are: OPEN, CLOSE, GET CHARACTERS, PUT CHARACTERS, GET RECORD, PUT RECORD, GET STATUS and SPECIAL. Other, device specific, commands are also supported by CIO and are described in section 5:Device specific information.

OPEN -- Assign device/filename to IOCB and ready for access.

Before a device/file may be accessed, it must be OPENed; this process links a specific IOCB to the appropriate device handler, initializes the device/file, initializes an CIO control variables, and passes device specific options to the device handler.

The following IOCB parameters are set by the user prior to calling CIO for an OPEN operation:

COMMAND BYTE = \$03

BUFFER ADDRESS = pointer to a device/filename specification (see section 5:Device/filename specification.)

AUX1 = OPEN direction bits, plus device dependent information.

AUX2 = device dependent information.

After an OPEN operation, CIO will have altered the following IOCB parameters:

HANDLER I.D. = index to the system Device Table; this is used only by CIO and must not be altered by the user.

DEVICE NUMBER = device number taken from the device/ filename specification and must not be altered by the user.

STATUS = result of OPEN operation; see Appendix B for a list of the possible status codes. In general, a negative status will indicate a failure to OPEN properly.

PUT ADDRESS = pointer to the PUT CHARACTERS routine for the device handler just OPENed. *** It is recommended that this pointer not be used ***

CLOSE -- Terminate access to device/file and release IOCB.

After the user is through accessing a given device/file, the CLOSE command is issued. This process completes any pending data writes, goes to the device handler for any device specific actions and then releases the IOCB.

The following IOCB parameter is set by the user prior to calling CIO:

COMMAND BYTE = \$0C

The following IOCB parameters are altered by CIO as a result of the CLOSE operation:

HANDLER I.D. = \$FF

STATUS = Result of CLOSE operation.

PUT ADDRESS = pointer to "IOCB not OPEN" routine.

GET CHARACTERS -- Read n characters (byte aligned access).

The specified number of characters are read from the device/file to the user supplied buffer. EOL characters have no termination feature: when using this function; there may be no EOL, or many EOLs, in the buffer after operation completion. There is a special case provided that passes a single byte of data in the 6502 A register when the buffer length is set to zero.

The following IOCB parameters are set by the user prior to calling CIO:

COMMAND BYTE = \$07

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = number of bytes to read; if this is zero, the data will be returned in the 6502 A register only.

The following IOCB parameters are altered by CIO as a result of the GET CHARACTERS operation:

STATUS = result of GET CHARACTERS operation.

BYTE COUNT/BUFFER LENGTH = number of bytes read to the buffer. The BYTE COUNT will always equal the BUFFER LENGTH except when an error or an end-of-file condition occurs.

PUT CHARACTERS -- Write n characters (byte aligned access).

The specified number of characters are written from the user supplied buffer to the device/file. EOL characters have no buffer terminating properties, although they have their standard meaning to the device/file receiving them; no EOLs are generated by CIO. There is a special case that allows a single character to be passed to CIO in the 6502 A register if the buffer length is zero.

The following IOCB parameters are set by the user prior to initiating the PUT CHARACTERS operation:

COMMAND BYTE = \$0B

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = number of bytes of data in buffer.

The following IOCB parameter is altered by CIO as a result of the PUT CHARACTERS operation:

STATUS = result of PUT CHARACTERS operation.

GET RECORD -- Read up to n characters (record aligned access).

Characters are read from the device/file to the user supplied buffer until either the buffer is full or an EOL character is read and put into the buffer. If the buffer fills before an EOL is read, CIO continues reading characters from the device/file until an EOL is read, then puts an EOL at the end of the buffer, and sets the status to indicate that a truncated record was read.

The following IOCB parameters are set by the user prior to calling CIO:

COMMAND BYTE = \$05

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = maximum number of bytes to read (including the EOL character).

The following IOCB parameters are altered by CIO as a result of the GET RECORD operation:

STATUS = result of GET RECORD operation.

BYTE COUNT/BUFFER LENGTH = number of bytes read to data buffer; this may be less than the maximum buffer length.

PUT RECORD -- Write up to n characters (record aligned access).

Characters are written from the user supplied buffer to the device/file until either the buffer is empty or an EOL character is written. If the buffer is emptied without writing an EOL character to the device/file, then CIO will send an EOL after the last user supplied character.

The following IOCB parameters are set by the user prior to calling CIO:

COMMAND BYTE = \$09

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = maximum number of bytes in buffer.

The following IOCB parameter is altered by CIO as a result of the PUT RECORD operation:

STATUS = result of PUT RECORD operation.

GET STATUS -- Return device dependent status bytes

The device controller is sent a STATUS command, and the controller returns four bytes of status information which are stored in DVSTAT [02EA]. See the subsections of 5.3 for the status information returned by each device.

The following IOCB parameters are set by the user prior to calling CIO:

COMMAND BYTE = \$0D

BUFFER ADDRESS = pointer to a device/filename specification if the IOCB is not already OPEN; see the discussion of the implied OPEN option below.

After a GET STATUS operation, CIO will have altered the following parameters:

STATUS = result of GET STATUS operation; see Appendix B for a list of the possible status codes.

DVSTAT = the four byte response from the device controller.

SPECIAL -- Special function

Any COMMAND BYTE value greater than \$0D is treated by CIO as a special case. Since CIO does not know what the function is, CIO transfers control to the device handler for complete processing of the operation.

The following IOCB parameters are set by the user prior to calling CIO:

COMMAND BYTE > \$0D

BUFFER ADDRESS = pointer to a device/filename specification if the IOCB is not already OPEN; see the discussion of the implied OPEN option below.

Other IOCB bytes may be setup, depending upon the specific SPECIAL command being performed.

After a SPECIAL operation, CIO will have altered the following parameters:

STATUS = result of SPECIAL operation; see Appendix B for a list of the possible status codes.

Other bytes may be altered, depending upon the specific SPECIAL command.

The device specific sections in 5.3 will detail the individual SPECIAL commands supported by the system.

Implied OPEN option

The GET STATUS and SPECIAL commands are treated specially by CIO; they may use an already OPEN IOCB to initiate the process or they may use an unOPENed IOCB. If the IOCB is unOPENed, then the BUFFER ADDRESS must contain a pointer to a device/filename specification, just as for the OPEN command; CIO will then OPEN that IOCB, perform the specified command and then CLOSE the IOCB again.

Device/filename specification

As part of the OPEN command, the IOCB buffer address parameter points to a device/filename specification, which is a string of ATASCII characters in the following format:

```
<specification> ::= <device>[<number>]:[<filename>]<eol>  
  
<device> ::= C|D|E|K|P|R|S  
<number> ::= 1|2|3|4|5|6|7|8  
<filename> has device dependent characteristics.  
<eol> ::= $9B
```

The following devices are supported at this writing:

```
C = Cassette drive  
D1 through D8 = Floppy diskette drives *  
E = Screen Editor  
K = Keyboard  
P = 40 column printer  
P2 = 80 column printer *  
R1 through R4 = RS-232-C interfaces *  
S = Screen display
```

Devices flagged by asterisks ('*') are supported by non-resident handlers.

If <number> is not specified, it is assumed to be 1.

The following examples show valid device/filename specifications:

```
C:           Cassette  
D2:BDAT     File "BDAT" on disk drive #2  
D:HOLD      File "HOLD" on disk drive #1  
K:           Keyboard
```

I/O example

The example provided in this section illustrates a simple example of an I/O operation using the CIO routine.

```
; This code segment illustrates the simple example of reading  
; text lines (records) from a disk file named "TESTER" on disk  
; drive #1. All symbols used are equated within the program  
; although many of the symbols are in the OS equate file.  
  
; The program performs the following steps:  
;  
; 1. OPENS the file 'D1:TESTER' using IOCB #3.  
; 2. Reads records until an error or EOF is reached.  
; 3. CLOSES the file.  
  
; I/O EQUATES
```

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

EOL= \$9B ; END OF LINE CHARACTER.
IOCB3= \$30 ; IOCB #3 OFFSET (FROM IOCB #0).

ICHID= \$0340 ; (HANDLER I. D. -- SET BY CIO).
ICDNO= ICHID+1 ; (DEVICE # -- SET BY CIO).
ICCOM= ICDNO+1 ; COMMAND BYTE.
ICSTA= ICCOM+1 ; STATUS BYTE -- SET BY CIO.
ICBAL= ICSTA+1 ; BUFFER ADDRESS (LOW).
ICBAH= ICBAL+1 ; BUFFER ADDRESS (HIGH).
ICPTL= ICBAH+1
ICPTH= ICPTL+1
ICBLL= ICPTH+1 ; BUFFER LENGTH (LOW).
ICBLH= ICBLL+1 ; BUFFER LENGTH (HIGH).
ICAX1= ICBLH+1 ; AUX 1.
ICAX2= ICAX1+1 ; AUX 2.

OPEN= \$03 ; OPEN COMMAND.
GETREC= \$05 ; GET RECORD COMMAND.
CLOSE= \$0C ; CLOSE COMMAND.

DREAD= \$04 ; OPEN DIRECTION = READ.
OWRIT= \$08 ; OPEN DIRECTION = WRITE.

EOF= \$88 ; END OF FILE STATUS VALUE.

CIOV= \$E455 ; CIO ENTRY VECTOR ADDRESS.

; FIRST INITIALIZE THE IOCB FOR FILE "OPEN".
;

ICCMD ~ LDX #10CB3 ; SETUP TO ACCESS IOCB #3.
LDA #OPEN ; SETUP OPEN COMMAND.
STA ICCOM, X

LDA #NAME ; SETUP BUFFER POINTER TO ...
STA ICBAL, X ; ... POINT TO FILENAME.
LDA #NAME/256
STA ICBAH, X

LDA #OREAD ; SETUP FOR OPEN READ.
STA ICAX1, X

LDA #0 ; CLEAR AUX 2.
STA ICAX2, X

; "OPEN" THE FILE.
;

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

```
JSR      CIOV          ; PERFORM "OPEN" OPERATION.  
BPL      TP10          ; YES -- STATUS WAS POSITIVE.  
  
JMP      ERROR         ; NO -- "OPEN" PROBLEM.  
  
; SETUP TO READ A RECORD.  
;  
TP10    LDA      #GETREC      ; SETUP "GET RECORD" COMMAND.  
STA      ICCOM, X  
  
LDA      #BUFF          ; SETUP DATA BUFFER POINTER.  
STA      ICBAL, X  
LDA      #BUFF/256       ;  
STA      ICBAH, X  
  
; READ RECORDS.  
;  
LOOP   LDA      #BUFFSZ      ; SETUP MAX RECORD SIZE ...  
STA      ICBLL, X        ; ... PRIOR TO EVERY READ.  
LDA      #BUFFSZ/256     ;  
STA      ICBLH, X  
  
JSR      CIOV          ; READ A RECORD.  
BMI      TP20          ; NO -- MAY BE END OF FILE.  
  
; A RECORD IS NOW IN THE DATA BUFFER "BUFF" IT IS TERMINATED BY  
; AN EOL CHARACTER, AND THE RECORD LENGTH IS IN "ICBLL" & "ICBLH".  
; THIS EXAMPLE WILL DO NOTHING WITH THE RECORD JUST READ.  
;  
JMP      LOOP          ; READ NEXT RECORD.  
  
; NEGATIVE STATUS ON READ -- CHECK FOR END OF FILE.  
;  
TP20    CPY      #EOF          ; END OF FILE STATUS?  
BNE      ERROR         ; NO -- ERROR.  
  
LDA      #CLOSE          ; YES -- CLOSE FILE.  
STA      ICCOM, X  
  
JSR      CIOV          ; CLOSE THE FILE.  
  
HLT      ""             ; *** END OF PROGRAM ***  
  
; DATA REGION OF EXAMPLE PROGRAM  
;
```

```
NAME      BYTE    "D1: TESTER", EOL
BUFFSZ= 80          ; 80 CHARACTER RECORD MAX
                     (INCLUDES EOL).
BUFF=   *           ; READ BUFFER.
*=     *+BUFFSZ
END
```

Device specific information

This section provides device specific information regarding device handlers that interface to CIO.

Keyboard handler (K:)

The Keyboard device is a read only device with a handler that supports the following CIO functions:

```
OPEN
CLOSE
GET CHARACTERS
GET RECORD
GET STATUS (null function)
```

The Keyboard handler may produce the following error statuses:

```
$80 -- [BREAK] key abort.
$88 -- End-of-file (produced by pressing CTRL-Z).
```

The Keyboard handler is one of the resident handlers, and therefore has a set of device vectors starting at location E420, as described further in section 5.

The keyboard can produce any of the 256 codes in the ATASCII character set as shown in Appendix F. Note that a few of the keyboard keys do not generate data at the Keyboard handler level; these keys are described below:

[/\!] - The ATARI key toggles a flag which enables/disables the inversion of bit-7 of each data character read. The Screen Editor editing keys are exempted from such inversion, however.

CAPS - The CAPS key provides three functions:
SHIFT-CAPS -- Alpha caps lock.
CTRL-CAPS -- Alpha CTRL lock.
CAPS -- Alpha unlock.

The system powersup and [S/RESET]s to the Alpha caps lock option.

Some key combinations are ignored by the handler, such as [CTRL]-4 through [CTRL]-9, [CTRL]-0, [CTRL]-1, [CTRL]-/ and all key combinations in which the [SHIFT] and [CTRL] keys are depressed simultaneously.

The [CTRL]-3 key generates an EOL character and returns EOF status.

The [BREAK] key generates an EOL character and returns BREAK status.

CIO function descriptions

The device specific characteristics of the standard CIO functions (described earlier in this section) are detailed below:

OPEN

The device name is 'K', and the handler ignores any device number and filename specification, if included.

There are no device dependent option bits in AUX1 or AUX2.

CLOSE

No special handler actions.

GET CHARACTERS and GET RECORD

The handler returns the ATASCII key codes to CIO as they are entered, with no facility for editing.

GET STATUS

The handler does nothing but set the status to \$01.

Theory of operation.

Everytime a keyboard key is pressed, an IRQ interrupt is generated as discussed in section 6 and is vectored to the Keyboard handler's interrupt service routine as shown later in section 6. The key code for the key pressed is then read and stored in data base variable CH [02FC]; this occurs whether or not there is an active read request to the Keyboard handler, thus effecting a one byte FIFO for keyboard entry. See section 4 (E8) for a discussion of the auto-repeat feature.

Whenever there is an active read request for the Keyboard handler, the handler monitors the CH variable for not containing the value \$FF (empty state). When CH shows non-empty, the handler takes the key code from CH and sets CH to \$FF again. The key code byte obtained from CH is not ATASCII code and has the following form:

7 0
+-+-+---+-+
|C|S| key code |
+-+-+---+-+--

Where: C = 1 if the CTRL key is pressed.
S = 1 if the SHIFT key is pressed.
The remaining 6 bits are the hardware key code.

The key code obtained is then converted to ATASCII using the first of the following rules which applies:

1. Ignore the code if the C & S bits are both set.
2. If the C bit is set, process the key as a CTRL code.
3. If the S bit is set, process the key as a SHIFT code.
4. If CTRL lock is in effect, process alpha characters as CTRL codes, all others as lower case.
5. IF SHIFT lock is in effect, process alpha characters as SHIFT codes, all others as lower case.
6. Else, process as lower case character.

Then, if the resultant code is not a Screen Editor control code, and if the video invert flag is set, set bit-7 of the ATASCII code (causes inverse video when displayed).

The keycode to ATASCII conversion table is shown on the next page. See also Appendix F.

KEYCODE TO ATASCII CONVERSION TABLE

Key Code	Key Cap	1. c.	SHIFT	CTRL	Key Code	Key Cap	1. c.	SHIFT	CTRL
00	L	6C	4C	0C	20	,	2C	5B	00
01	J	6A	4A	0A	21	SPACE	20	20	20
02	;	3B	3A	7B	22		2E	5D	60
03	--	--	--	--	23	N	6E	4E	0E
04	--	--	--	--	24	--	--	--	--
05	K	6B	4B	0B	25	M	6D	4D	0D
06	+	2B	5C	1E	26	/	2F	3F	--
07	*	2A	5E	1F	27) ! (--	--	--
08	O	6F	4F	0F	28	R	72	52	12
09	--	--	--	--	29	--	--	--	--
0A	P	70	50	10	2A	E	65	45	05
0B	U	75	55	15	2B	Y	79	59	19
0C	RET	9B	9B	9B	2C	TAB	7F	9F	9E
0D	I	69	49	09	2D	T	74	54	14
0E	-	2D	5F	1C	2E	W	77	57	17
0F	=	3D	7C	1D	2F	Q	71	51	11
10	V	76	56	16	30	9	39	28	--
11	--	--	--	--	31	--	--	--	--
12	C	63	43	03	32	0	30	29	--
13	--	--	--	--	33	7	37	27	--
14	--	--	--	--	34	BACKS	7E	9C	FE
15	B	62	42	02	35	8	38	40	--
16	X	78	58	18	36	<	3C	7D	7D
17	Z	7A	5A	1A	37	>	3E	9D	FF
18	4	34	24	--	38	F	66	46	06
19	--	--	--	--	39	H	68	48	08
1A	3	33	23	9B*	3A	D	64	44	04
1B	6	36	26	--	3B	--	--	--	--
1C	ESC	1B	1B	1B	3C	CAPS	--	--	--
1D	5	35	25	--	3D	G	67	47	07
1E	2	32	22	FD	3E	S	73	53	13
1F	1	31	21	--	3F	A	61	41	01

* [CTRL]-3 returns EOF status.

The inverse of this table (ATASCII to keystroke) is given in Appendix F.

Display handler (S:)

The Display device is a read/write device with a handler that supports the following CIO functions:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS (null function)

DRAW
FILL

The Display handler may produce the following error statuses:

\$84 -- Invalid special command.
\$8D -- Cursor out of range.
\$91 -- Screen mode > 11.
\$93 -- Not enough memory for screen mode selected.

The Display handler is one of the resident handlers, and therefore has a set of device vectors starting at location E410.

Screen modes

The display screen may be operated in any of 20 configurations (modes 1 through 8, with or without split screen, plus modes 0 and 9 through 11 without split screen). Mode 0 is the text displaying mode and modes 1 through 11 are all different graphics modes (although modes 2 and 3 do display a subset of the ATASCII character set). Modes 9 through 11 require a GTIA chip to be installed in place of the standard CTIA chip.

TEXT MODE (mode 0)

In text mode the screen is physically comprised of 24 lines of 40 characters per line; however, the display area is limited by program alterable left and right margins which default to 2 and 39 (of a possible 0 and 39).

A program controllable cursor shows the destination of the next character to be output. The cursor is visible as the inverted video representation of the current character at the destination position.

The text screen data is organized internally as variable length logical lines; when the screen is cleared, the internal representation is 24 empty lines. As text is sent to the screen, each EOL marks the end of a logical line; or if more than 3 physical lines of text are sent, a logical line will be formed every 3 physical lines. The number

of physical lines used to comprise a logical line (1 to 3) is always the minimum required to hold the data for that logical line.

The text screen "scrolls" upward whenever a text line at the bottom row of the screen extends past the right margin or a text line at the bottom row is terminated by an EOL. Scrolling has the effect of removing the entire logical line that starts at the top of the screen and then moving all subsequent lines upward to fill in the void. The cursor will also move upward if the logical line deleted exceeds one physical line.

All data going to or coming from the text screen is represented in 8 bit ATASCII code as shown in Appendix E.

Text Modes 1 and 2

In text modes 1 and 2 the screen is physically comprised of either 24 lines of 20 characters (mode1) or 12 lines of 20 characters (mode 2). The left and right margins are of no consequence in these modes and there is no visible cursor. There are no logical lines associated with the data and in all regards these modes are treated as graphics modes by the handler.

Data going to or coming from the screen is in the form shown below:

7	0
C	D

Where: C is the color/data set select field

C	Color Value	Color (default)	Character Register (see appendix K)	Character Set CHBAS=\$E0	Character S CHBAS=\$E2
0	green	P(PFI)			
1	gold	(PFO)	' !-' ?'		
2	gold	(PFO)	' '-' '		
3	green	(PF1)	' '-' '		
4	red	(PF3)	' '-' '		
5	blue	(PF2)	' '-' '		
6	blue	(PF2)	' '-'		
7	red	(PF3)	' '-'		

D is a 5 bit truncated ATASCII code which selects the specific character within the set selected by the C field. See Appendix E for the graphics representations of the characters.

Database variable CHBAS [02F4] allows for the selection of either of two data sets. The default value of \$E0 provides the capital letters, numbers and punctuation characters; the alternate value

of \$E2 provides lower case letters and the special character graphics set.

GRAPHICS MODES (modes 3 through 11)

The screen has varying physical characteristics for each of the graphics modes as shown in Appendix H. Depending upon the mode, a 1 to 16 color selection is available for each pixel and the screen size varies from 20 by 12 (lowest resolution) to 320 by 192 (highest resolution) pixels.

There is no visible cursor for the graphics mode output.

Data going to or coming from the graphics screen is represented as 1 to 8 bit codes as shown in Appendix H and in the GET/PUT diagrams following.

SPLIT SCREEN CONFIGURATIONS

In split screen configurations, the bottom of the screen is reserved for four lines of mode 0 text. The text region is controlled by the Screen Editor, and the graphics region is controlled by the Display handler. Two cursors are maintained in this configuration so that the screen segments may be managed independently.

In order to operate in split screen mode, the Screen Editor must first be OPENed and then the Display handler must be OPENed using a separate IOCBL (with the split screen option bit set in AUX1).

CIO function descriptions

The device specific characteristics of the standard CIO functions (described earlier in this section) are detailed below:

OPEN

The device name is 'S', and the handler ignores any device number and filename specification, if included.

The handler supports the following options:

	7	0
AUX1	! C!S!W!R!	!
	-----	-----

Where: C = 1 indicates to inhibit screen clear on OPEN.

S = 1 indicates to setup a split screen configuration (for modes 1 through 8 only).

R & W are the direction bits (read & write).

```
7          0
+---+---+---+---+
AUX2    |           ! mode   !
+---+---+---+---+
```

Where: mode is the screen mode (0 through 11).

Note: If the screen mode selected is 0, then the AUX1 options are ignored.

Because the Display handler dynamically allocates high address memory for use in generating the screen display, and because different amounts of memory are needed for the different screen modes, the Display handler and the user must share memory utilization information. Prior to initiating an OPEN command the variable APPMHI [000E] should contain the highest address of RAM needed by the user; the Screen handler will OPEN the screen only if no RAM is needed at or below that address.

Upon return from a screen OPEN, the variable MEMTOP [02E5] will contain the address of the last free byte at the end of RAM memory prior to the screen required memory.

As a result of every OPEN command, the following screen variables are altered:

The text cursor is enabled (CRSINH = 0).
The tabs are set to the default settings (2 & 39).
The color registers are set to the default values.
(shown in Appendix H). Tabs are set at positions 7, 15, 23, 31, 39,
47, 55, 63, 71, 79, 87, 95, 103, 111, 119.

CLOSE

No special handler actions.

GET CHARACTERS and GET RECORD

Returns data in the following screen mode dependent forms, where each byte contains the data for one cursor position (pixel); there is no facility for having the handler return packed graphics data.

lspace 1;need 20

```
7          0
+---+---+---+---+
|     ATASCII    |      Mode 0
+---+---+---+---+

+---+---+---+---+
| C | D |      | Modes 1,2 -- C = color, ata set
+---+---+---+---+ select.
                                         D = truncated ATASCII.
```

```
+-----+  
| zero | D | Modes 3, 5, 7 -- D = color.  
+-----+  
  
+-----+  
| zero | DI | Modes 4, 6, 8 -- D = color.  
+-----+  
  
+-----+  
| zero | D | Modes 9, 10, 11 -- D = data.  
+-----+
```

As each data byte is returned, the cursor is moved to the next cursor position. For mode 0, the cursor will stay within the specified margins; for all other modes, the margins are ignored.

PUT CHARACTERS and PUT RECORD

The handler accepts display data in the following screen mode dependent forms; there is no facility for the handler to receive graphics data in packed form.

```
7 0  
+-----+  
| ATASCII | Mode 0  
+-----+  
  
+-----+  
| C | D | * Modes 1, 2 -- C = color,  
+-----+ D = truncated ATASCII.  
  
+-----+  
| ? | D | Modes 3, 5, 7 -- D = color.  
+-----+  
  
+-----+  
| ? | DI | Modes 4, 6, 8 -- D = color.  
+-----+  
  
+-----+  
| ? | D | Modes 9, 10, 11 -- D = data.  
+-----+
```

NOTE: For all modes, if the output data byte equals \$9B (EOL) that byte will be treated as an EOL character; and if the output data byte equals \$7D (CLEAR) that byte will be treated as a screen clear character.

As each data byte is written, the cursor is moved to the next cursor position. For mode 0, the cursor will stay within the specified margins; for all other modes, the margins are ignored.

While outputting, the Display handler monitors the keyboard to detect the pressing of the [CTRL]-1 key combination; when this occurs, the

handler loops internally until that key combination is pressed again, thus effecting a stop/start function to freeze the screen display. Note that there is no ATASCII code associated with either the [CTRL]-1 key combination or the start/stop function; the stop/start function may be controlled only from the keyboard (or by altering database variable CH as discussed in Appendix K)

GET STATUS

No handler action except to set the status to \$01.

DRAW

This special command draws a simulated "straight" line from the current cursor position to the location specified in ROWCRS [0054] and COLCRS [0055]. The color of the line is taken from the last character processed by the Display handler or Screen Editor. To force the color, store the desired value in ATACHR [02FB]. At the completion of the command, the cursor will be at the location specified by ROWCRS and COLCRS.

The value for the command byte for DRAW is \$11.

FILL

This special command fills an area of the screen defined by two lines with a specified color. The command is setup the same as in DRAW, but as each point of the line is drawn, the routine scans to the right performing the procedure shown below (in Pascal notation):

```
WHILE PIXEL [ROW,COL] = 0 DO
  BEGIN
    PIXEL [ROW,COL] := FILDAT;
    COL := COL + 1;
    IF COL > Screen right edge THEN COL := 0
  END;
```

An example of a FILL operation is shown below:

```
+ 1
3 +-----+
      . -----
      . -----
      .
4 +-----+
      .
+ 2
```

Where: '-' represents the fill operation.
' ' are the line points, with '+' for the endpoints.

- 1 -- set cursor and plot point.
- 2 -- set cursor and DRAW line.

3 -- set cursor and plot point.
4 -- set fill data value, set cursor and FILL.

FILDAT [02FD] contains the fill data and ROWCRS and COLCRS contain the cursor coordinates of the line endpoint. The value in ATACHR [02FB] will be used to draw the line; ATACHR always contains the last data read or written, so if the steps above are followed exactly, ATACHR will not have to be modified.

The value for the command byte for FILL is \$12.

User alterable database variables

Certain functions of the Display handler require the user to examine and/or alter variables in the OS database; the paragraphs that follow describe some of the more commonly used handler variables. There are additional descriptions to be found in Appendix K B1-55.

CURSOR POSITION

The cursor position for the graphics screen or mode 0 text screen is maintained in two variables: ROWCRS [0054], the display row number, and COLCRS [0055], the display column number. Both numbers range from 0 to the maximum number of rows/columns - 1. The cursor may be set outside of the defined text margins with no ill effect; this region may be read from and written to when the cursor is controlled by the user. The home position (0,0) for both text and graphics is the upper left corner of the screen.

ROWCRS is a single byte, and COLCRS is two bytes with the least significant byte being at the lower address.

When these variables are altered by the user, the screen representation of the cursor will not move until the next I/O operation involving the display is performed.

INHIBIT/ENABLE VISIBLE CURSOR DISPLAY

The user may inhibit the display of the text cursor on the screen by setting the variable CRSINH [02F0] to any non-zero value. Subsequent I/O will not generate a visible cursor.

The user may enable the display of the text cursor by setting CRSINH to zero. Subsequent I/O will then generate a visible cursor.

TEXT MARGINS

As mentioned earlier, the text screen has user alterable left and right margins, which are normally set to 2 and 39 by the OS. The variable LMARGN [0052] defines the left margin and RMARGN [0053] defines the right margin. The leftmost margin value is 0 and the rightmost margin value is 39.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

The margin values inclusively define the useable portion of the screen for all operations in which the user does not explicitly alter the cursor location variables as described prior to this paragraph.

COLOR CONTROL

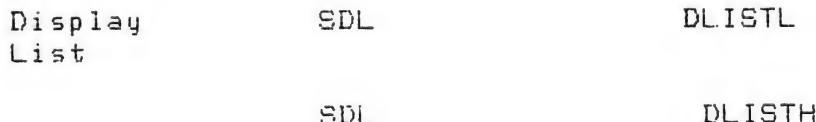
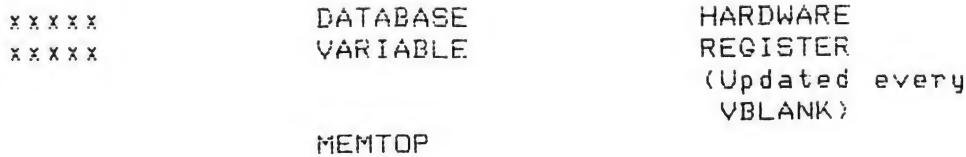
As part of normal stage 2VBLANK processing (as discussed in Section 6) the hardware color registers are updated using data from the OS database. Shown below are the database variable names, the hardware register names and the function of each register; see Appendix H for the mode dependent uses for the registers.

Database	Hardware	Function
COLOR0	COLPFO	PF0 -- Playfield 0.
COLOR1	COLPF1	PF1 -- Playfield 1.
COLOR2	COLPF2	PF2 -- Playfield 2.
COLOR3	COLPF3	PF3 -- Playfield 3.
COLOR4	COLBK	BAK -- Playfield background.
PCOLR0	COLPM0	PM0 -- Player/missile 0.
PCOLR1	COLPM1	PM1 -- Player/missile 1.
PCOLR2	COLPM2	PM2 -- Player/missile 2.
PCOLR3	COLPM3	PM3 -- Player/missile 3.

Theory of operation

The Display handler automatically sets up all memory resources required to create and maintain the screen display at OPEN time. The screen generation hardware requires that two distinct data areas exist for graphics modes: 1) a display list and 2) a screen data region; a third data area must exist for text modes which defines the screen representation for each of the text characters. The ATARI personal computer HARDWARE MANUAL must be referenced for a complete understanding of the material that is to follow.

The simplified block diagram below shows the relationships between the memory and hardware registers used to setup a screen display (without player/missile objects) by the O.S.; be aware that the hardware allows for many other possibilities.



Screen Data SAVMSC

Graphics
and/or end of RAM memory
text

Char.	E000	CHBAS=EO	CHBASE	
Defs.				Specials and numbers E000
in OS	E3FF		Capital Letters E100	
ROM				Special Graphics E200
		Color Color COLPFO		
		Color 1 regs. COLPF2		
		Color 2 COLPF3		
		Color 3 COLBK	Lower case E300	
		Color 4	Letters	

In the preceding diagram the following relationships are present:

Database variables SDL/STL/SDL/STH contain the address of the current display list; as part of the Stage 1 VBlank process this address is stored in the hardware display list address registers DLISLT and DLISTH.

The display list itself defines the characteristics of the screen to be displayed and points to the memory containing the data to be displayed.

Database variable CHBAS contains the msb of the base address of the character representations for the character data (text modes only). The default value for this variable is \$E0, which declares that the character representations start at memory address E000 (the character set provided by the O.S. in ROM). Each character is defined as an 8x8 bit matrix, requiring 8 bytes per character; since a character code contains up to 7 significant bits (set of 128 characters), 1024 bytes are required to define the largest set. The O.S. ROM contains the default set in the region from E000 to E3FF.

All Character codes are converted by the handler from ATASCII to an internal code and vice versa, as shown below:

ATASCII CODE	INTERNAL CODE
00-1F	40-5F
20-3F	00-1F

40-5F	20-3F
60-7F	60-7F
80-9F	CO-DF
A0-BF	80-9F
CO-DF	A0-BF
EO-FF	EO-FF

The character set in ROM is layed out in internal code order. The reason for the internal code being different from the external code (ATASCII) is based upon three considerations: 1) ATASCII codes for all but the special graphics characters were to be similar to ASCII, specifically the alphabetic, numeric, and punctuation character codes are identical to ASCII, 2) in text modes 1 and 2 it was desired that one haracter subset included capital letters, numbers and punctuation and the other character subset include lower case letters and special graphics characters and 3) the codes for the capital and lower case letters were to be identical in text modes 1 and 2.

Database variables COLOR0 through COLOR4 contain the current color register assignments; these are also stored in the hardware color registers aspart of the stage 1 VBLANK process, thus providing synchronized color changes. Appendix H provides more information regarding the color registers.

Database variable SAVMSC points to the lowest memory address of the screen data region, which corresponds to the data displayed at the upper left corner of the display.

When the display handler receives an OPEN command, it first determines the screen mode from the OPEN IOCB. It then allocates memory from the end of RAM (as specified by database variable RAMTOP) downward; first for the screen data and then for the display list. If thereis sufficient memoryavailable, the screen data region is cleared, the display list is created, and the display list address is stored to the database.

Screen Editor (E:)

The Screen Editor is a read/write handler that uses the Keyboard handler and the Display handler to provide "line at a time" input with interactive editing functions, as well as formatted output.

The Screen Editor supports the following CIO functions:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS

PUT RECORD
GET STATUS (null function)

The Screen Editor may produce the following error statuses:

see Keyboard handler and Display handler.

The Screen Editor is one of the resident handlers, and therefore has a set of device vectors starting at location E400.

The Screen Editor may be thought of as a program which reads key data from the Keyboard handler and sends each character to the Display handler for immediate display; and in addition, accepts data from the user to send to the Display handler. In addition, the Screen Editor reads data from the Display handler (not the Keyboard handler) for the user. In fact, the Keyboard handler, Display handler and the Screen Editor are all contained in one monolithic hunk of code, and thus, are even more closely related than indicated.

Most of the behaviors already defined for the Keyboard handler and the Display handler apply to the Screen Editor, so the discussions in this section will pertain to deviations from those behaviors or to additional features that are part of the Screen Editor only. The Screen editor deals only with text data (screen mode 0) as described in section 5. A split screen configuration is allowed which is also explained.

Whereas the Display handler allows the graphics and text screens to be readable on program demand, the Screen Editor gives the operator at the keyboard the control of what portion of the screen is to be read and when it is to be read. The choice of when is governed by the [RETURN] key, and the choice of where is governed by the location of the cursor when the [RETURN] key is pressed. When the [RETURN] key is pressed, the entire logical line within which the cursor resides is then made available to the calling program. Trailing blanks in a logical line are never returned as data, however. After all of the data in the line has been sent to the caller (this may entail multiple READ CHARACTERS functions if desired), the cursor is positioned to the beginning of the logical line following the one just read.

CIO function descriptions

The device specific characteristics of the standard CIO functions (described earlier in section 5) are detailed below:

OPEN

The device name is 'E', and the Screen Editor ignores any device number and filename specification, if included.

The Screen Editor supports the following option:

7	0
+---+---+---+---+---+	
AUX1	I IWIRI IFI
+---+---+---+---+---+	

Where: R & W are the direction bits (read and write).

F = 1 indicates that a "forced read" is desired (see GET CHARACTER and GET RECORD for more information).

CLOSE

No special handler actions.

GET CHARACTER and GET RECORD

Normally the Screen Editor will return data to the caller only when prompted to do so by having the operator at the keyboard press the [RETURN] key. However, the "forced read" OPEN option, allows a caller to read text data without operator intervention; when a read operation is commanded, the Screen Editor will return data from the start of the logical line in which the text cursor is located and then move the cursor to the beginning of the following logical line. A read of the last logical line on the screen will cause the screen data to scroll.

A special case occurs when characters are output without a terminating EOL and then additional characters are appended to that logical line from the keyboard. When the [RETURN] key is pressed, only the keyboard entered characters are sent to the caller, unless the cursor has been moved out of and then back into the logical line, in which case all of the logical line will be sent.

PUT CHARACTER and PUT RECORD

The handler accepts ATASCII characters as one character per byte. Sixteen of the 256 ATASCII characters are control codes; the EOL code has universal meaning, but most of the other control codes have special meaning only to a display or print device. The Screen Editor processing of the ATASCII control codes is explained below:

CLEAR (\$7D) -- The current display is cleared of all data and the cursor is placed at the home position (upper left corner of the screen).

CURSOR UP (\$1C) -- The cursor is moved up by one physical line. The cursor will wrap from the top line of the display to the bottom line.

CURSOR DOWN (\$1D) -- The cursor is moved down by one physical line. The cursor will wrap from the bottom line of the display to the top line.

CURSOR LEFT (\$1E) -- The cursor is moved left by one column. The cursor will wrap from the left margin of a line to the right margin of the same line.

CURSOR RIGHT (\$1F) -- The cursor is moved right by one column. The cursor will wrap from the right margin of a line to the left margin of the same line.

BACKSPACE (\$7E) --- The cursor is moved left by one column (but never past the beginning of a logical line) and the character at that new position is changed to a blank (\$20).

SET TAB (\$9F) -- A tab point is established at the logical line position at which the cursor is residing. The logical line tab position is not synonymous with the physical line column position since the logical line may be up to 3 physical lines in length. For example, tabs may be set at the 15th, 30th, 45th, 60th and 75th character positions of a logical line as shown below:

0 2 9 19 29 39 Screen column #.
--L-----+-----+-----+-----R L/R = margins.

x x -----T-----T----- A logical line.
x x -----T-----T-----T- x = inaccessible
x x ----- columns.

Note the effect of the left margin in defining the limits of the logical line.

The handler default tab settings are shown below:

0 2 9 19 29 39 Screen column #.
--L---+-----+-----+-----R L/R = margins.

x x T-----T-----T-----T-----T-----T A logical line.
x x -----T-----T-----T-----T-----T x = inaccessible
x x -----T-----T-----T-----T-----T columns.

CLEAR TAB (\$9E) -- The current cursor position within the logical line is cleared from being a tab point. There is no "clear all tab points" facility provided by the handler.

TAB (\$7F) -- The cursor is moved to the next tab point in the current logical line, or to the beginning of the next line if no tab point is found. Note that this function will not increase the logical line length to accommodate a tab point outside the current length (e.g. the logical line length is 38 characters and there is a tab point at position 50).

INSERT LINE (\$9D) -- The physical line in which the cursor resides, and all physical lines below that line, are moved down by one physical line; the last logical line on the display may be truncated as a result. The blank physical line at the insert point becomes the beginning of a new logical line. A logical line may be split into two logical lines by this process, the last half of the original logical line being concatenated with the blank physical line formed at the insert point.

DELETE LINE (\$9C) -- The logical line in which the cursor resides is deleted and all data below that line is moved upward to fill the void. Empty logical lines are created at the bottom of the display.

INSERT CHARACTER (\$FF) -- The character at the cursor position, and all remaining characters in the logical line, are moved one position to the right and the character at the cursor position is set to blank. The last character of the logical line will be lost when the logical line is full and a character is inserted. The number of physical lines comprising a logical line may increase as a result of this function.

DELETE CHARACTER (\$FE) -- The character on which the cursor resides is removed, and the remainder of the logical line to the right of the deleted character is moved to the left by one position. The number of physical lines comprising a logical line may decrease as a result of this function.

ESCAPE (\$1B) -- The next non-EOL character following this code is displayed as data, even if it would normally be treated as a control code. The sequence ESC ESC will cause the second ESC character to be displayed.

BELL (\$FD) -- An audible tone is generated; the display is not modified.

END OF LINE (\$9B) -- In addition to its record termination function, the EOL causes the cursor to advance to the beginning of the next logical line. When the cursor reaches the bottom line of the screen, the receipt of an EOL will cause the screen data to scroll upward by one logical line.

Output start/stop using the [CTRL]-1 key is processed as explained in section 5:Display handler (S:).

GET STATUS

The handler takes no action other than to set the status to \$01.

User alterable database variables

See also the Display handler database variable discussion.

CURSOR POSITION (split screen)

When in a split screen configuration, ROWCRS and COLCRS are associated with the graphics portion of the display and two other variables, TXTROW [0290] and TXTCOL [0291], are associated with the text window. TXTROW is a single byte, and TXTCOL is two bytes with the least significant byte being at the lower address. Note that the most significant byte of TXTCOL should always be zero.

The home position (0,0) for the text window is the upper left corner of the window.

ENABLE/INHIBIT OF CONTROL CODES IN TEXT

Normally all text mode control codes are operated upon as received, but sometimes it is desireable to have the control codes displayed as if they were data characters. This is done by setting the variable DSPFLG [02FE] to any non-zero value before outputting the data containing control codes. Setting DSPFLG to zero restores normal processing of text control codes.

Cassette handler (C:)

The Cassette device is a read or write device with a handler that supports the following CIO functions:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS (null function)

The Cassette handler may produce the following error statuses:

\$80 -- [BREAK] key abort.
\$84 -- Invalid AUX1 byte on OPEN.
\$88 -- End-of-file.
\$8A-90 -- SIO error set (see Appendix C).

The Cassette handler is one of the resident handlers, and therefore has a set of device vectors starting at location E440.

CIO function descriptions

The device specific characteristics of the standard CIO functions are detailed below:

OPEN

The device name is 'C', and the handler ignores any device number and filename specification, if included.

The handler supports the following option:

AUX2	7	0
	+---+---+---+---+	
	:C:	:
	+---+---+---+---+	

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

Where: C = 1 indicates that the cassette is to be read/written without stop/start between records (continuous mode).

When the cassette is OPENed for input, a single audible tone is generated, using the keyboard speaker, as a prompt for the operator to verify that the cassette player is setup for reading (power on, Serial Bus cable connected, tape cued to start of file and PLAY button depressed). When the cassette is ready, the operator may press any keyboard key (except [BREAK]) to initiate tape reading.

When the cassette is OPENed for output, two closely spaced audible tones are generated, using the keyboard speaker, as a prompt for the operator to verify that the cassette player is setup for writing (as above, plus REC button depressed). When the cassette is ready, the operator may press any keyboard key (except [BREAK]) to initiate tape writing. Note that there is no way for the computer to verify that the REC button (or even the PLAY button) is depressed, so it is possible for the file not to be written, with no immediate indication of this fact.

There is a potential problem with the cassette in that when the cassette is OPENed for writing, the motor keeps running until the first record (128 data bytes) is written. If 128 data bytes are written or the cassette is CLOSEd within about 30 seconds of the OPEN, and no other serial bus I/O is performed, then there is no problem. However, if those conditions are not met, some noise will be written to the tape prior to the first record and an error will occur when that tape file is read later. If lengthy delays are anticipated between the time the cassette file is OPENed and the time that the first cassette record (128 data bytes) is written, then a dummy record should be written as part of the file; typically 128 bytes of some innocuous data would be written, such as all zeroes, all \$FFs or all blanks (\$20).

The system will sometimes emit whistling noises after cassette I/O has occurred. The sound can be eliminated by storing \$03 to SKCTL [D20F], thus bring POKEY out of the two-tone (FSK) mode.

CLOSE

The CLOSE of a tape read stops the cassette motor.

The CLOSE of a tape write does the following:

- Writes any remaining user data in the buffer to tape.
- Writes an End-of-file record.
- Stops the cassette motor.

GET CHARACTERS and GET RECORD

The handler returns data in the following format:

```
7          0
+-+-+---+---+---+
|   data byte   |
+-+-+---+---+---+
```

PUT CHARACTERS and PUT RECORD

The handler accepts data in the following format:

```
7          0
+-+-+---+---+---+
|   data byte   |
+-+-+---+---+---+
```

The handler attaches no significance to the data bytes written, a value of \$9B (EOL) causes no special action.

GET STATUS

The handler does no more than set the status to \$01.

Theory of operation.

The cassette handler writes and reads all data in fixed length records of the format shown below:

```
+-+-+---+---+---+
|0 1 0 1 0 1 0 1|      Speed measurement bytes.
+-+-+---+---+---+
|0 1 0 1 0 1 0 1|
+-+-+---+---+---+
| control byte   |
+-+-+---+---+---+
|     128         |
=     data         =
|     bytes        |
+-+-+---+---+---+
|     checksum     |      (Managed by SIO, not the
+-+---+---+---+    handler).
```

The control byte contains one of three values:

\$FC indicates the record is a full data record (128 bytes).

\$FA indicates the record is a partially full data record; fewer than 128 bytes were supplied by the user. This case may occur only in the record prior to the End-of-file. The number of user supplied data bytes in the record is contained in the byte prior to the checksum.

\$FE indicates the record is an End-of file record; the data portion is all zeroes for an End-of-file record.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

The checksum is generated and checked by the SIO routine and is part of the tape record, but is not contained in the handler's record buffer CASBUF [03FD].

The processing of the speed measurement bytes during cassette reading is discussed in section 4:Central database description.

File structure

The Cassette handler writes a file to the cassette device with a file structure that is totally imposed by the handler (soft format). A file consists of the following three elements:

- A 20 second leader of mark tone.
- Any number of data record frames.
- End-of file frame.

The cassette frames referred to above are formatted as shown below:

```
frame = pre-record write tone (PRWT),  
       + data record,  
       + post-record gap (PRG)
```

The non-data portions of a frame have characteristics which are dependent upon the write OPEN mode, i.e. continuous or start/stop.

Stop/start PRWT = 3 seconds of mark tone.
Continuous PRWT = .25 seconds of mark tone.

Stop/start PRG = up to 1 second of unknown tones.
Continuous PRG = from 0 to n seconds of unknown tones, where n is dependent upon user program timing.

The inter-record gap (IRG) between any two records will thus consist of the PRG of the first record followed by the PRWT of the second record.

Printer handler (P:)

The Printer device is a write only device with a handler that supports the following CIO functions:

```
OPEN  
CLOSE  
PUT CHARACTERS  
PUT RECORD  
GET STATUS
```

The Printer handler may produce the following error statuses:

8A-90 -- SIO error set (see Appendix C).

The Printer handler is one of the resident handlers, and therefore has a set of device vectors starting at location E430.

CIO function descriptions

The device specific characteristics of the standard CIO functions are detailed below:

OPEN

The device name is 'P', and the handler ignores any device number and filename specification, if included.

CLOSE

The handler writes any data remaining in its buffer to the printer device, with trailing blanks to fill out the line.

PUT CHARACTERS and PUT RECORD

The handler accepts print data in the following format:

7	0
+-+-+-+---+-+	
ATASCII	
+-+-+-+---+-+	

The only ATASCII control code of any significance to the handler is the EOL character. The printer device ignores bit-7 of every data byte and prints a sub-set of the remaining 128 codes, see Appendix G for the printer character set.

The handler supports the following print option:

7	0
+-+-+-+---+-+	
AUX2	print mode
+-+-+-+---+-+	

Where: \$4E ('N') selects normal printing (40 chars per line).
\$53 ('S') selects sideways printing (29 chars per line).
\$57 ('W') selects wide printing (not supported by printer device.).
Any other value (including 00) is treated as a normal (N) print select, without producing an error status.

GET STATUS

The handler obtains a four byte status from the printer controller and puts it in system location DVSTAT [02EA]. The format of the status bytes is shown below:

```
+---+---+---+---+  
| command stat. | DVSTAT + 0  
+---+---+---+---+  
| AUX2 of prev. | + 1  
+---+---+---+---+  
| timeout | + 2  
+---+---+---+---+  
| (unused) | + 3  
+---+---+---+---+
```

The command status contains the following status bits:

Bit-0 indicates an invalid command frame was received.

Bit-1 indicates an invalid data frame was received.

Bit-7 indicates an intelligent controller (normally = 0).

The next byte contains the AUX2 value from the previous operation.

The timeout byte contains a controller provided maximum timeout value (in seconds)

Theory of operation.

The ATARI B20TM Printer is a line at a time printer, rather than a character at a time printer, so the user data must be buffered by the handler and sent to the device in records corresponding to one print line (40 characters for normal, 29 characters for sideways).

The printer device does not attach any significance to the EOL character, so the handler does the appropriate blank fill whenever it sees an EOL.

Disk File Manager (D:)

The File Management Subsystem (FMS) includes a disk bootable (RAM resident) DFM which maintains a collection of named files on diskettes. Up to 4 disk drives (D1: through D4:) may be accessed, and up to 64 files per diskette may be accessed; the system disks supplied by ATARI allow a single disk drive (D1) and up to 3 OPEN files, but these numbers may be altered by the user as described later in this section. The Disk File Manager supports the following CIO functions:

OPEN FILE
OPEN DIRECTORY
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS

NOTE
POINT
LOCK
UNLOCK
DELETE
RENAME
FORMAT

The Disk File Manager may produce the following error statuses:

\$03 -- Last data from file (EOF on next read).
\$88 -- End-of-file.
\$8A-90 -- SIO error set (see Appendix C).
\$A0 -- Drive number specification error.
\$A1 -- No sector buffer available (too many open files).
\$A2 -- Disk full.
\$A3 -- Fatal I/O error in directory or bitmap.
\$A4 -- Internal file # mismatch (structural problem).
\$A5 -- File name specification error.
\$A6 -- Point information in error.
\$A7 -- File locked to this operation.
\$A8 -- Special command invalid.
\$A9 -- Directory full (64 files).
\$AA -- File not found.
\$AB -- Point invalid (file not OPENed for update).

CIO function descriptions

The device specific characteristics of the standard CIO functions are detailed below:

OPEN FILE

The device name is 'D' and up to 4 disk drives may be accessed (D1 through D4); the disk filename may be from 1 to 8 characters in length with an optional 1 to 3 character extension.

The OPEN FILE command supports the following options:

7	0
+-+-+---+-+---+-+	
AUX1	!W!R! !A!
+-+-+---+-+---+-+	

Where: W & R are the direction bits.

WR = 00 is invalid

01 indicates OPEN for read only.

10 indicates OPEN for write only.

11 indicates OPEN for read/write (update).

A = 1 indicates appended output when W = 1.

The various valid AUX1 options are now explained.

OPEN input (AUX1 = \$04)

The indicated file is OPENed for input. Any wild card characters are used to search for the first match. If the file is not found, an error status is returned, and no file will be OPENed.

OPEN output (AUX1 = \$08)

The indicated file is OPENed for output starting with the first byte of the file, if the file is not locked. Any wild card characters are used to search for the first match. If the file already exists, the existing file will be DELETED before OPENing the named file as a new file. If the file does not already exist, it will be created.

A file OPENed for output will not appear in the directory until it has been CLOSED. If an output file is not properly CLOSED, some or all of the sectors that were acquired for it may be lost until the disk is reformatted.

A file that is OPENed for output may not be OPENed concurrently for any other access.

OPEN append (AUX1 = \$09)

The indicated file is OPENed for output starting with the byte after the last byte of the existing file (which must already exist), if the file is not locked. Any wild card characters are used to search for the first match.

If a file OPENed for append is not properly CLOSED, the appended data will be lost, the existing file will remain unmodified and some or all of the sectors that were acquired for the appended portion may be lost until the disk is reformatted.

OPEN update (AUX1 = \$0C)

The indicated file (which must already exist) will be OPENed for update provided it is not locked. Any wild card characters are used to search for the first match.

The GET, PUT, NOTE and POINT operations are all valid, and may be intermixed as desired.

If a file OPENed for update is not properly CLOSED, a sector's worth of information may be lost to the file. A file OPENed for update may not be extended.

Device/filename specification

The handler expects to find a device/filename specification of the following form:

D[<number>]:<filename><EOL>

where:

<number> ::= 1|2|3|4
<filename> ::= [<primary>][. [<extension>]]<terminator>
<primary> ::= an upper case alpha character followed by 0 to 7 alphanumeric characters. If the primary name is less than 8 characters, it will be padded with blanks; if it is greater than 8 characters, the extra characters will be ignored.
<extension> ::= Zero to 3 alphanumeric characters. If the extension name is missing or less than 3 characters, it will be padded with blanks; if it is greater than 3 characters, the extra characters will be ignored.
<terminator> ::= <EOL><blank>

The following are all valid device/filenames for the disk:

D1:GAME.SRC
D:MANUAL6
D:.WHY
D3:FILE.
D4:BRIDGE.002

Filename wildcarding

The filename specification may be further generalized to include the use of the "wildcard" characters '*' and '?'. These wildcard characters allow portions of the primary and/or extension to be abbreviated as follows:

The '?' character in the specification allows any file name character at that position to produce a "match". For example, WH? will match files named WHO, WHY, WH4, etc., but not a file named WHAT.

The '*' character causes the remainder of the primary or extension field in which it is used to be effectively padded with '?' characters. For example, WH* will match WHO, WHEN, WHATEVER, etc.

Some valid uses of wildcard specifications are shown below:

*. SRC	Files having an extension of SRC.
BASIC.*	Files named BASIC with any extension.
.	All files.
H*.?	Files beginning with H and having a 0 or 1 character extension.

If wildcarding is used with an OPEN FILE command, the first file found (if any) that meets the specification will be the one (and only one) opened.

OPEN DIRECTORY

The OPEN DIRECTORY command allows the user to read directory information for the selected filename(s), using normal GET CHARACTERS or GET RECORD commands. The information read will be formatted as ATASCII records, suitable for printing, as shown below. Wildcarding may be used to obtain information for multiple files or the entire disk.

The OPEN DIRECTORY command uses the same CIO parameters as a standard OPEN FILE command:

COMMAND BYTE = \$03

BUFFER ADDRESS = pointer to device/filename specification.

AUX1 = \$06

After the directory is OPENed, a record will be returned to the caller for each file that matches the OPEN specification. The record, which contains only ATASCII characters, is formatted as shown below:

1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8
+--+	--+	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-
s b	primary name	ext b count e	+--+	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-

Where: s = '*' or ' ', with '*' indicating file locked.

b = blank.

primary name = left justified name with blank fill.

ext = left justified extension with blank fill.

b = blank.

count = number of sectors comprising the file.

e = EOL (\$9B).

After the last filename match record is returned, an additional record is returned, which indicates the number of unused sectors available on the disk. The format for this record is shown below:

/space 1;need 5

1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	
+--+	--+	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-	-+-

```
|count| F R E E   S E C T O R S |  
+---+---+---+---+---+---+---+---+---+
```

Where: count = the number of unused sectors on the disk.
e = EOL (\$9B).

The EOF statuses (\$03 and \$88) are returned as per a normal data file when the last directory record is read.

The OPENing of another disk file while the directory read is OPEN will cause subsequent directory reads to malfunction, so care must be taken to avoid this situation.

CLOSE

On closing a file read, the handler releases all internal resources being used to support that file.

On closing a file write, the handler writes any residual data from its file buffer for that file to the disk, updates the directory and allocation map for the associated disk, and releases all internal resources being utilized to support that file.

GET CHARACTERS and GET RECORD

Characters are read from the disk and passed to CIO as a raw data stream; none of the ATASCII control characters have any special significance. A status of \$03 is returned when the last byte of file is returned and a status of \$88 is returned if an attempt is made to read past the last byte.

PUT CHARACTERS and PUT RECORD

Characters are obtained from CIO and written to the disk as a raw data stream; none of the ATASCII control characters have any special significance.

GET STATUS

The indicated file is checked and one of the following status byte values is returned in ICSTA and register Y:

\$01 -- File found & unlocked.
\$A7 -- File locked.
\$AA -- File not found.

Special CIO functions

The DFM supports a number of SPECIAL commands, which are device specific; these are explained in the paragraphs that follow.

NOTE (COMMAND BYTE = \$25)

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

This command returns to the caller the exact disk location of the next byte to be read or written, in the variables shown below:

ICAX3 = l. s. b. of the disk sector number.
ICAX4 = m. s. b. of the disk sector number.
ICAX5 = relative sector displacement to byte (0-124).

POINT (COMMAND BYTE = \$26)

This command allows the user to specify the exact disk location of the next byte to be read or written. In order to use this command, the file must have been OPENed with the "update" option.

ICAX3 = l. s. b. of the disk sector number.
ICAX4 = m. s. b. of the disk sector number.
ICAX5 = relative sector displacement to byte (0-124).

LOCK

This command allows the user to prevent write access to any number of named files. Locked files may not be deleted, renamed nor opened for output unless they are first unlocked. Locking a file that is already locked is a valid operation. The handler expects a device/filename specification; then all occurrences of the filename specified will be locked, using the wildcard rules.

The following IOCB parameters are setup by the user prior to calling CIO:

COMMAND BYTE = \$23

BUFFER ADDRESS = pointer to device/filename specification.

After a LOCK operation, the following IOCB parameter will have been altered:

STATUS = result of LOCK operation; see Appendix B for a list of possible status codes.

UNLOCK

This command allows the user to remove the lock status of any number of named files. Unlocking a file that is not locked is a valid operation. The handler expects a device/filename specification; then all occurrences of the filename specified will be unlocked, using the wildcard rules.

The following IOCB parameters are setup by the user prior to calling CIO:

COMMAND BYTE = \$24

BUFFER ADDRESS = pointer to device/filename specification.

After an UNLOCK operation, the following IOCB parameter will have been altered:

STATUS = result of UNLOCK operation; see Appendix B for a list of possible status codes.

DELETE

This command allows the user to delete any number of unlocked named files from the directory of the selected disk and to deallocate the disk space used by the files involved. The handler expects a device/filename specification; then all occurrences of the filename specified will be deleted, using the wildcard rules.

The following IOCB parameters are setup by the user prior to calling CIO:

COMMAND BYTE = \$21

BUFFER ADDRESS = pointer to device/filename specification.

After a DELETE operation, the following IOCB parameter will have been altered:

STATUS = result of DELETE operation; see Appendix B for a list of possible status codes.

RENAME

This command allows the user to change the filenames of any number of unlocked files on a single disk. The handler expects to find a device/filename specification as shown below:

<device spec>:<filename spec>,<filename spec><EOL>

All occurrences of the first filename will be replaced with the second filename, using the wildcard rules. No protection is provided against forming duplicate names, and once formed, duplicate names cannot be separately renamed or deleted; however, an OPEN FILE command will always select the first file found that matches the filename specification, so that file will always be accessible. The RENAME command does not alter the content of the files involved, merely the name in the directory.

Examples of some valid RENAME name specifications are shown below:

D1:*, SRC, *, TXT
D: TEMP, FDATA
D2: F*, F*. OLD

The following IOCB parameters are setup by the user prior to calling CIO:

COMMAND BYTE = \$20

BUFFER ADDRESS = pointer to device/filename specification.

After a RENAME operation, the following IOCB parameter will have been altered:

STATUS = result of RENAME operation; see Appendix B for a list of possible status codes.

FORMAT

This command allows the user to physically format a diskette, which is required before the diskette can be used to store data; the physical formatting process writes a new copy of every sector on the "soft sectored" diskette, with the data portion of each sector containing all zeroes. When the physical formatting process is complete, the FMS creates an initial Volume Table of Contents (VTOC) and an initial File Directory; as part of this process the boot sector (#1) is permanently reserved. The result of the FORMAT process will be the creation of an "empty" non-system disk.

The following IOCB parameters are setup by the user prior to calling CIO:

COMMAND BYTE = \$FE

BUFFER ADDRESS = pointer to device specification.

After a FORMAT operation, the following IOCB parameter will have been altered:

STATUS = result of FORMAT operation; see Appendix B for a list of possible status codes.

To create a system disk, a copy of the boot file must next be written to sectors #2-n. This is accomplished by writing the file named 'DOS.SYS', which is a name that is recognized by the FMS even though it is not in the directory initially.

Theory of operation

The resident OS initiates the disk boot process, as described in section 10.2, by reading disk sector #1 to memory and then transferring control to the "boot continuation address" (boot address + 6). The boot continuation program contained in sector #1 then continues to load the remainder of the File Management Subsystem to memory using additional information contained in sector #1. The File Management Subsystem loaded will contain a Disk File Manager and, optionally, a Disk Utilities (DOS) package.

When the boot process is complete, the Disk File Manager will allocate additional RAM for the creation of sector buffers. Sector buffers are allocated based upon information in the boot record as shown below:

Byte 9 = maximum number of OPEN files; one buffer per (the maximum value is 8).

Byte 10 = drive select bits; one buffer per (1-4 only).

The Disk File Manager will then insert the name 'D' and the handler vector table address in the Device Table.

NOTE: There is a discrepancy between the Disk File Manager's numbering of disk sectors (0-719) and the disk controller's numbering of disk sectors (1-720); as a result, only sectors 1- 719 are used by the Disk File Manager.

The Disk File Manager uses the Disk handler to perform all disk reads and writes; the DFM's function is to support and maintain the directory/file(bitmap structures as described in the following pages:

FMS DISK UTILIZATION

The map below shows the disk sector utilization for a standard 720 sector diskette.

+-----+	
BOOT record	Sector 1
+-----+	
FMS BOOT	Sector 2 -+
= file =	
'DOS.SYS'	Sector n +- Note 1
+-----+	
User	Sector n+1 -+
= File =	
Area	Sector 359 (\$167)
+-----+	
VOTC(note 2)	Sector 360 (\$168)
+-----+	
File	Sector 361 (\$169)
= Directory =	
	Sector 368 (\$170)
+-----+	
User	
= File =	
Area	Sector 719 (\$20F)
+-----+	
unused	Sector 720 (\$2D0)
+-----+	

NOTE 1 - If the diskette is not a system diskette, then the User File Area starts at sector 2 and no space is reserved for the FMS BOOT file.

Note 2 -- VOTC stands for volume table of contents.

FMS BOOT RECORD FORMAT

The FMS BOOT record (sector #1) is a special case of disk booted software as described in section 10.2. The format for the FMS BOOT record is shown below:

+-----+		
boot flag = 0	Byte 0	
+-----+		
# sectors = 1	1	
+-----+		
boot address	2	
+-----+		
= 0700		
+-----+		
init address	4	
+-----+		
+-----+		
JMP = \$4B	6	
+-----+		
boot read		
+ continuation +		
address		
+-----+ -----+		
max files = 3	9 Note 1	
+-----+ -----+		
drive bits = 1	10 Note 2	
+-----+ -----+		
alloc direc = 0	11 Note 3	
+-----+ -----+		
boot image end		
+-----+ -----+		
address + 1		FMS configuration data
+-----+ -----+		
boot flag < 0	14 Note 4	
+-----+ -----+		
sector count	15 Note 5	
+-----+ -----+		
'DOS.SYS'		
+ starting +		
sector number		
+-----+ -----+		
code for 2nd		
phase of boot		

NOTE 1 - Byte 9 specifies the maximum number of concurrently OPEN files to be supported. This value may range from 1 to 8.

NOTE 2 - Byte 10 specifies the specific disk drive numbers to be supported using a bit encoding scheme as shown below:

```
7 6 5 4 3 2 1 0
+-+-+-+---+---+-
|           |4|3|2|1| where a 1 indicates a selected drive.
+-+-+-+---+---+-
```

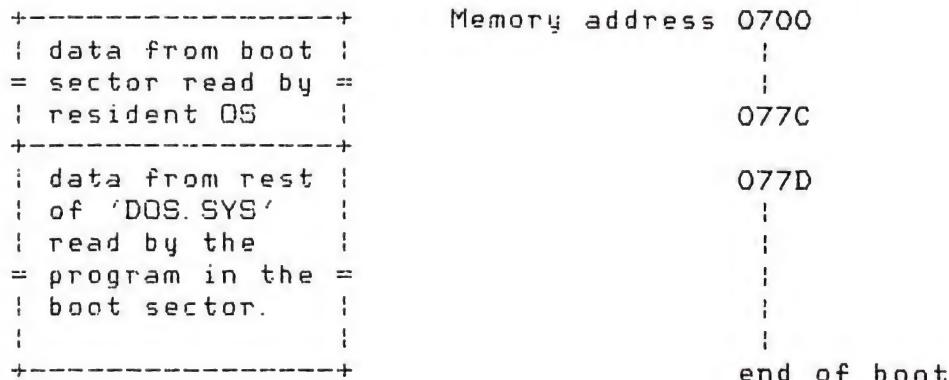
NOTE 3 - Byte 11 specifies the buffer allocation direction, this byte should equal 0.

NOTE 4 - Byte 14 must be non-zero for the second phase of the boot process to initiate; this flag indicates that the file 'DOS.SYS' has been written to the disk.

NOTE 5 - This byte is assigned as being the sector count for the 'DOS.SYS' file, but is in actuality an unused byte.

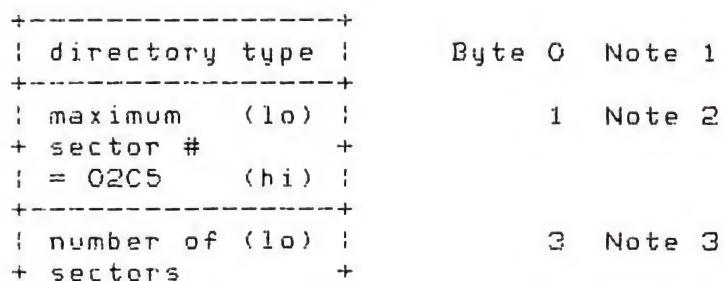
BOOT PROCESS MEMORY MAP

The diagram below shows how the boot sector (part of file DOS.SYS) and subsequent sectors are loaded to memory as part of the boot process.



VOLUME TABLE OF CONTENTS (VTOC)

The format for the FMS volume table of contents (VTOC, sector 360) is shown in the diagram below:



```
| .available (hi) |
+-----+
|           |
=           =
|           |
+-----+
|           |          10
= volume bit map =
|           |
+-----+
|           |
=           =
|           |
+-----+
```

Where the volume bit map is organized as shown below:

7	0	
+---+---+---+---+---+---+		
1 2 3 4 5 6 7	Byte 10 of VTOC	
+---+---+---+---+---+---+		
8 9	11	
= =		
	99	
+---+---+---+---+---+---+		

At each map bit position, a 0 indicates the corresponding sector is in use and a 1 indicates that the sector is available.

NOTE 1 - The directory type byte must equal 0.

NOTE 2 - The maximum sector number is apparently not used because it is incorrectly set to 709 decimal although the true maximum sector number is 719, for the DFM.

NOTE 3 - The number of sectors available is initially set to 709 after a diskette is freshly formatted; this number is adjusted as files are created and deleted to show the number of sectors available. The sectors which are initially reserved are 1 and 360-368.

FILE DIRECTORY FORMAT

There are eight sectors (361-368) reserved for a file directory, each sector containing directory information for up to eight files, thus providing for a maximum of 64 files for any volume. The format of a single 16 byte file entry is shown below:

flag byte	Byte 0
sector (lo)	1
count	
(hi)	
starting (lo)	3
sector	
number (hi)	
(1)	5
(2)	
(3)	
file (4)	
name (5)	
primary (6)	
(7)	
(8)	
file (1)	13
name (2)	
extension (3)	

Where the flag byte has the following bits assigned:

Bit-7 = 1 if the file has been deleted.
Bit-6 = 1 if the file is in use.
Bit-5 = 1 if the file is locked.
Bit-0 = 1 if OPEN output.

The flag byte may take on the following values:

\$00 = entry not yet used (no file).
\$40 = entry in use (normal CLOSED file).
\$41 = entry in use (OPEN output file).
\$60 = entry in use (locked file).
\$80 = entry available (prior file deleted).

Sector count is the number of sectors comprising the file.

FMS FILE SECTOR FORMAT

The format of a sector in a user's data file is shown below:

7	0
+---+---+---+---+---+---+	
data	+0
+---+---+---+---+---+---+	
file #	hi +125
+---+---+---+---+---+---+	
forward pointer	+126
+---+---+---+---+---+---+	
S! byte count	+127
+---+---+---+---+---+---+	

The file # is a redundant piece of information which is used to verify file integrity; the file number field contains the value of the directory position of that file. If there is ever a mismatch between the file's position in the directory and the file number as contained in each sector, the Disk File Manager will generate the error \$A4.

The forward pointer field contains the ten bit value of the disk sector number of the next sector of the file. The pointer will equal zero for the last sector of a file.

The S bit indicates whether or not the sector is a "short sector" (one containing fewer than 125 data bytes). S is equal to one when the sector is short.

The byte count field contains the number of data bytes in the sector.

Non CIO I/O

Some portions of the I/O subsystem may or must be accessed independently of the Central I/O Utility (CIO); this section discusses those areas.

Resident device handler vectors

All of the OS ROM resident device handlers may be accessed via sets of vectors which are part of the OS ROM. The primary reason for using these vectors would be to increase the speed of I/O operations which utilize fixed device assignments, such as output to the Display handler. For each resident handler there is a set of vectors ordered as shown below:

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

+-----+	
+- OPEN -+	+0
+-----+	
+- CLOSE -+	+2
+-----+	
+- GET BYTE -+	+4
+-----+	
+- PUT BYTE -+	+6
+-----+	
+- GET STATUS -+	+8
+-----+	
+- SPECIAL -+	+10
+-----+	
+- JMP -+	+12
+- INIT -+	
+-----+	

See section 9 for a detailed description of the data interface for each of these handler entry points.

Each of the vectors contains the address (lo,hi) of the handler entry point minus one, so a technique similar to the one shown below is required to access the desired routines:

```
VTBASE=$E400 ; BASE OF VECTOR TABLE.  
  
LDX #xx ; OFFSET TO DESIRED ROUTINE.  
LDA data  
JSR GOVEC ; SEND DATA TO ROUTINE.  
. .  
LDX #yy ; OFFSET TO DIFFERENT ROUTINE.  
JSR GOVEC ; GET DATA FROM ROUTINE.  
STA data  
  
GOVEC TAY ; SAVE REGISTER A.  
LDA VTBASE+1,X ; ADDRESS M. S. B. TO STACK.  
PHA  
LDA VTBASE,X ; ADDRESS L. S. B. TO STACK.  
PHA  
TYA ; RESTORE REGISTER A.  
RTS ; JUMP TO ROUTINE.
```

The JMP INIT slot in each set of vectors jumps to the handler initialization entry (not minus one).

The base address of the vector set for each of the resident handlers is shown below:

Screen Editor (E:)	E400.
Display handler (S:)	E410.
Keyboard handler (K:)	E420.

Printer handler (P:) E430.
Cassette handler (C:) E440.

The resident disk handler is not CIO compatible, so its interface does not use a vector set; the disk handler interface is discussed in section 5.

Resident Disk handler

The resident Disk handler (not to be confused with the Disk File Manager) is responsible for all physical accesses to the disk. The unit of data transfer for this handler is a single disk sector containing 128 data bytes.

Communication between the user and the Disk handler is effected using the system's Device Control Block (DCB), which is also used for handler/SIO communication as described in section 9. The DCB is twelve bytes long, in which some bytes are user alterable and some are for use by the Disk handler and/or the Serial I/O Utility (SIO). The user supplies the required DCB parameters and then does a JSR DSKINV [E453].

Each of the DCB bytes will now be described, and the system equate file name for each will be given.

SERIAL BUS I.D. -- DDEVIC [0300]

This byte is setup by the Disk handler to contain the Serial Bus I.D. for the drive to be accessed, and is not user alterable.

DEVICE NUMBER -- DUNIT [0301]

This byte is setup by the user and contains the disk drive number to be accessed (1 - 4).

COMMAND BYTE -- DCOMND [0302]

This byte contains the disk device command to be performed and is setup by the user.

STATUS BYTE -- DSTATS [0303]

This byte contains the status of the command upon return to the caller. See Appendix C for a list of the possible status codes.

BUFFER ADDRESS -- DBUFLO [0304] & DBUFHI [0305]

This two byte pointer contains the address of the source or destination of the disk sector data. For the disk status command, the user need not supply an address; the Disk handler will obtain the status and insert the address of the status buffer in this field.

DISK TIMEOUT VALUE -- DTIMLO [0306]

This timeout value (in whole seconds) is supplied by the handler for use by SIO.

BYTE COUNT -- DBYTLO [0308] & DBYTHI [0309]

This two byte counter indicates the number of bytes transferred to or from the disk as a result of the most recent command, and is setup by the handler.

SECTOR NUMBER -- DAUX1 [030A] & DAUX2 [030B]

This two byte number specifies the disk sector number (0 - 719) to read or write. DAUX1 contains the least significant byte, and DAUX2 contains the most significant byte.

Disk handler commands

There are five commands supported by the Disk handler:

GET SECTOR
(PUT SECTOR ---*** not supported by handler ***)
PUT SECTOR WITH VERIFY
STATUS REQUEST
FORMAT DISK

GET SECTOR (Command byte = \$52)

The handler reads the specified sector to the user's buffer and returns the operation status. The following DCB parameters are set by the user prior to calling the Disk handler:

COMMAND BYTE = \$52.

DEVICE NUMBER = disk drive number (1-4).

BUFFER ADDRESS = pointer to user's 128 byte buffer.

SECTOR NUMBER = sector number to read.

Upon return, several of the other DCB parameters will have been altered, however, the STATUS BYTE will be the only one of interest to the user.

PUT SECTOR (Command byte = \$50)

*** Not supported by current handler ***

The handler writes the specified sector from the user's buffer and returns the operation status. The following DCB parameters are set by the user prior to calling the Disk handler:

COMMAND BYTE = \$50.

DEVICE NUMBER = disk drive number (1-4).

BUFFER ADDRESS = pointer to user's 128 byte buffer.

SECTOR NUMBER = sector number to write.

Upon return, several of the other DCB parameters will have been altered, however, the STATUS BYTE will be the only one of interest to the user.

PUT SECTOR WITH VERIFY (Command byte = \$57)

The handler writes the specified sector from the user's buffer and returns the operation status. This command differs from PUT SECTOR in that the disk controller reads the sector data after writing to verify the write operation. (byte by byte compare???) Aside from the COMMAND BYTE value, the calling sequence is identical to PUT SECTOR.

STATUS REQUEST (Command byte = \$53)

The handler obtains a four byte status from the disk controller and puts it in system location DVSTAT [02EA]. The operation status format is shown below:

7	0
+++---++-+-+--+	
command stat.	DVSTAT + 0
++-+-+--+--+-+--	
hardware stat.	+ 1
++-+-+--+--+-+--	
timeout	+ 2
++-+-+--+--+-+--	
(unused)	+ 3
++-+-+--+--+-+--	

The command status contains the following status bits:

Bit-0 = 1 indicates an invalid command frame was received.

Bit-1 = 1 indicates an invalid data frame was received.

Bit-2 = 1 indicates that a PUT operation was unsuccessful.

Bit-3 = 1 indicates that the disk is write protected.

Bit-4 = 1 indicates active/standby.

The hardware status byte contains the status register of the INS1771-1 Floppy Disk Controller chip used in the disk controller. See the documentation for that chip for information relating to the meaning of each bit in the byte.

The timeout byte contains a controller provided maximum timeout value (in seconds) to be used by the handler.

The following DCB parameters are set by the user prior to calling the Disk handler:

COMMAND BYTE = \$53.

DEVICE NUMBER = disk drive number (1-4).

Upon return, several of the other DCB parameters will have been altered, however, the STATUS BYTE will be the only one of interest to the user.

FORMAT DISK (Command byte = \$21)

The handler commands the disk controller to format the entire disk and then to verify it. All bad sector numbers, up to a maximum of 63, are returned and put in the supplied buffer, followed by two bytes of all ones (\$FFFF). The following DCB parameters are set by the user prior to calling the Disk handler:

COMMAND BYTE = \$21.

DEVICE NUMBER = disk drive number (1-4).

BUFFER ADDRESS = pointer to user's 128 byte buffer.

Upon return, the following DCB parameters will be of interest to the user:

STATUS BYTE = status of operation.

BYTE COUNT = number of bytes of bad sector information in user's buffer, not including the \$FFFF terminator. If there are no bad sectors, the count will equal zero.

Educational System Program Cassettes

Educational System Program Cassette tapes are recorded in 1/4 track stereo format at 1 7/8 inches per second. The tape can be recorded in both directions, where tracks 1 and 2 are side A left and right; and tracks 3 and 4 are side B right and left (industry standard). On each side, the left channel (1 or 4, outside tracks) is used for audio and the right channel (2 or 3, inside tracks) is used for digital information.

The audio channel is recorded in the normal manner. The digital channel is recorded using FSK encoding and the recording is asynchronous byte, with no record structure.

All data bits on the tape (not including start/stop) are stored in bit inverted form; that is, all zero bits are stored as one bits and vice versa.

ATARI EDUCATIONAL SYSTEM CHARACTER SET

HEX	CHAR	NOTES
00	Null	Reserved for Atari use--see text.
01	SOH	Dorsett screen hold character--see text.
02	STX	Inhibit marked text.
03	ETX	Clear Screen
04	EOT	Stop Tape
05	ENQ	Enable marked text
06	ACK	
07	BEL	End of tape
08	BS	Select left response
09	HT	Select middle response
0A	LF	Select right response
0B	VT	Select any response
0C	FF	Reserved for Dorsett
0D	CR	Carriage return
0E	SO	Reserved for Dorsett
0F	SI	Reserved for Dorsett
10	DLE	
11	DC1	Border Brown (note: colors may not match this description.)
12	DC2	Border red
13	DC3	Border Orange
14	DC4	Border Yellow
15	NAK	Border Green
16	SYN	Border Blue
17	ETB	
18	CAN	
19	EM	Text background brown
1A	SUB	Text Background red
1B	ESC	Text background orange
1C	FS	Text background yellow
1D	GS	Text background green
1E	RS	Text background blue
1F	US	
20		Space
21	!	
22	"	
23	Pi	
24	\$	
25	%	
26		Overline
27	/	
28	(
29)	
2A		Wheel
2B	+	
2C	,	
2D	-	
2E	.	
2F	/	
30		Centered period
31	1	
32	2	

33 3
34 4
35 5
36 6
37 7
38 8
39 9
3A :
3B ;
3C <
3D =
3E >
3F ?
40
41 A
42 B
43 C
44 D
45 E
46 F
47 G
48 H
49 I
4A J
4B K
4C L
4D M
4E N
4F O
50 P
51 Q
52 R
53 S
54 T
55 U
56 V
57 W
58 X
59 Y
5A Z
5B "Head"
5C \
5D "Torso"
5E ^ (actually somewhat lowered)
5F
60 Double parallel vertical slashes
61 a
62 b
63 c
64 d
65 e
66 f
67 g
68 h

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM. CO16555

59	i	
6A	j	
6B	k	
6C	l	
6D	m	
6E	n	
6F	o	
70	p	
71	q	
72	r	
73	s	
74	t	
75	u	
76	v	
77	w	
78	x	
79	y	
9 7A		
7B	"Re-entrant" (dotted) slash	
7C	"Re-entrant" (dotted) backslash	
7C	Right-justified vertical slash	
7D	n-tilde (for spanish)	
7F	three parallel horizontal lines	

NOTE: All characters above are the same character but 'flagged' (see text) if the top bit is set, except that the flag bit is ignored on all control characters except the following (flag is not allowed on NULL):

99	EM	Inset Brown
9A	SUB	Inset Red
9B	ESC	Inset Orange
9C	FS	Inset Yellow
9D	GS	Inset Green
9E	RS	Inset Blue

Serial bus I/O (SIO)

Input/Output to devices other than the keyboard, the screen and the ATARI controller port devices must utilize the Serial I/O bus. This bus contains data, control and clock lines to be used to allow the computer to communicate with external devices on this "daisy chained" bus. Every device on the bus has a unique identifier and will respond only when directly addressed.

The resident system provides a Serial I/O Utility (SIO), which provides a standardized high-level program interface to the bus. SIO is utilized by the resident Disk, Printer and Cassette handlers and is intended to be used by non-resident handlers (as described in section 9) or by applications, as well. For a detailed description of the program/SIO interface and for a detailed bus specification refer to section 9.

Device characteristics

This section describes the physical characteristics of the devices that interface to the ATARI 400 and ATARI 800 Personal Computer

Systems. Where applicable, data capacity, data transfer rate, storage format, SIO interface and cabling will be detailed.

Keyboard

The keyboard input rate is limited by the OS keyboard reading procedure to be 60 characters per second. The code for each key is shown in Table ??, section 5. A picture of the ATARI 400 Personal Computer System keyboard is shown on the following page. The keyboard hardware has no buffering and is rate limited by the debounce algorithm used.

Display

The television screen display generator has many capabilities that are not used by the Display handler (as described in section 5 and shown in Appendix H); there are additional display modes, object generators, hardware display scrolling and many other features which are described in the ATARI Personal Computer System HARDWARE MANUAL.

Since all display data is stored in RAM, the display data update rate is limited primarily by the software routines that generate and format the data and access the RAM. The generation of the display from the RAM is accomplished by the ANTIC and CTIA chips using Direct Memory Access (DMA) to access the RAM data.

The internal storage formats for display data for the various modes are detailed in the ATARI Personal Computer System HARDWARE MANUAL.

ATARI 410 Program Recorder

The ATARI 410 Program Recorder has the following characteristics:

DATA CAPACITY:

xx characters per C-60 tape (unformatted).
xx characters per C-60 tape (formatted, continuous).
xx characters per C-60 tape (formatted, stop/start).

DATA TRANSFER RATES:

xx characters per second (unformatted).
xx characters per second, average (formatted, stop/start).

STORAGE FORMAT:

Tapes are recorded in 1/4 track stereo format at 1 7/8 inches per second. The tape can be recorded in both directions, where tracks 1 and 2 are side A left and right; and tracks 3 and 4 are side B right and left (industry standard). On each side, the left channel (1 or 4)

is used for audio and the right channel (2 and 3) is used for digital information.

The audio channel is recorded the normal way. The digital channel is recorded using the POKEY two-tone mode producing FSK data at up to 600 baud. The MARK frequency is 5327 Hz and the SPACE frequency is 3995 Hz. The transmission of data is asynchronous byte serial as seen from the computer; POKEY reads or writes a bit serial FSK sequence for each byte, in the following order:

```
1 start bit (SPACE)
data bit-0 +-+
data bit-1   |
             +- 0 = SPACE, 1 = MARK.
data bit-6   |
data bit-7 +-+
1 stop bit (MARK)
```

The only control the computer has over tape motion is motor start/stop; and this only if the PLAY button is pressed by the user. In order for recording to take place, the user must press both the REC and PLAY buttons on the cassette. The computer has no way to sense the position of these buttons, nor even if a 410 is cabled to the computer, so the user must be careful when using this device.
SIO INTERFACE

The cassette device utilizes portions of the serial bus hardware, but does not follow any of the protocol as defined in section 9.

ATARI 820 Printer

The ATARI 820 printer has the following characteristics:

DATA CAPACITY:

40 characters per line (normal printing)
29 characters per line (sideways printing)

DATA TRANSFER RATES:

Bus rate: xx characters per second.
Print time (burst): xx characters per second.
Print time (average): xx characters per second.

STORAGE FORMAT:

3 7/8 inch wide paper.
5X7 dot matrix, impact printing.

Normal format --
40 characters per line.

6 lines per inch (vertical).
12 characters per inch (horizontal).

Sideways format --
29 characters per line.
6 lines per inch (vertical).
9 characters per inch (horizontal).

SIO INTERFACE

The controller serial bus I.D. is \$40.

The controller supports the following SIO commands (see section 5 for more information regarding the handler and section 9 for a general discussion of bus commands):

GET STATUS

The computer sends a command frame of the format shown below:

Device I.D. = \$40.
Command byte = \$53.
Auxilliary 1 = doesn't matter.
Auxilliary 2 = doesn't matter.
Checksum = checksum of bytes above.

The printer controller responds with a data frame of the format shown in earlier in this section as part of the GET STATUS discussion.

PRINT LINE

The computer sends a command frame of the format shown below:

Device I.D. = \$40.
Command byte = \$57.
Auxilliary 1 = doesn't matter.
Auxilliary 2 = \$4E for normal print or \$53 for sideways.
Checksum = checksum of bytes above.

The computer sends a data frame of the format shown below:

Leftmost character of line (column 1).
Next character of line (column 2).

Rightmost character of line (column 40 or 29).
Checksum byte.

Note that the data frame size is variable, either 41 or 30 bytes in length, depending upon the print mode specified in the command frame.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

ATARI 810 Disk Drive

The ATARI 810TM disk has the following characteristics:

DATA CAPACITY:

720 sectors of 128 bytes each (Disk handler format).
709 sectors of 125 data bytes each (Disk File Manager format).

DATA TRANSFER RATES:

Bus rate: xx characters per second.
Seek time: xx msec. per track + xx msec.
Rotational latency: xx msec maximum (xx rpm).

STORAGE FORMAT:

5 1/4 inch diskette, soft sectored by the controller.
40 tracks per diskette.
18 sectors per track.
128 bytes per sector.
Controlled by National INS1771-1 formatter/controller chip.
Sector interlace factor =

SIO INTERFACE

The controller serial bus I.D.s range from \$31 (for 'D1') to \$34 (for 'D4').

The controller supports the following SIO commands (see earlier in this section for information about the disk handler and section 9 for a general discussion of bus commands):

GET STATUS

The computer sends a command frame of the format shown below:

Device I.D. = \$31-34.
Command byte = \$53.
Auxilliary 1 = doesn't matter.
Auxilliary 2 = doesn't matter.
Checksum = checksum of bytes above.

The disk controller responds with a data frame of the format shown earlier in this section as part of the STATUS REQUEST discussion.

PUT SECTOR (WITH VERIFY)

The computer sends a command frame of the format shown below:

Device I.D. = \$31-34
Command byte = \$57
Auxilliary 1 = low byte of sector number.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

Auxilliary 2 = high byte of sector number (1-720).
Checksum = checksum of bytes above.

The computer sends a data frame of the format shown below:

128 data bytes.
Checksum byte.

The disk controller writes the frame data to the specified sector, then reads the sector and compares the content with the frame data. The COMPLETE byte value indicates the status of the operation.

PUT SECTOR (NO VERIFY)

The computer sends a command frame of the format shown below:

Device I. D. = \$31-34
Command byte = \$50.
Auxilliary 1 = low byte of sector number.
Auxilliary 2 = high byte of sector number (1-720).
Checksum = checksum of bytes above.

The computer sends a data frame of the format shown below:

128 data bytes.
Checksum byte.

The disk controller writes the frame data to the specified sector, then sends a COMPLETE byte value which indicates the status of the operation.

GET SECTOR

The computer sends a command frame of the format shown below:

Device I. D. = \$31-34
Command byte = \$52.
Auxilliary 1 = low byte of sector number.
Auxilliary 2 = high byte of sector number (1-720).
Checksum = checksum of bytes above.

The disk controller sends a data frame of the format shown below:

128 data bytes.
Checksum byte.

FORMAT DISK

The computer sends a command frame of the format shown below:

Device I. D. = \$31-34
Command byte = \$21.
Auxilliary 1 = doesn't matter.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

Auxilliary 2 = doesn't matter.
Checksum = checksum of bytes above.

The disk controller completely formats the disk (generates 40 tracks of 18 soft sectors per track with the data portion of each sector equal to all zeroes) and then reads each sector to verify its integrity. A data frame of 128 bytes plus checksum is returned in which the sector numbers of all bad sectors (up to a maximum of 63 sectors) are contained, followed by two consecutive bytes of \$FF. If there are no bad sectors on the disk the first two bytes of the data frame will contain \$FF

ATARI 850TM Interface Module

See ATARI 850 Interface Module Manual.

6 - INTERRUPT PROCESSING..

Introduction

There are three general interrupt types processed by the 6502 microcomputer: chip reset, non-maskable interrupts (NMI) and maskable interrupts (IRQ). The IRQ type may be enabled and disabled using the 6502 CLI and SEI instructions, whereas the NMI type may not be disabled at the processor level; the NMI interrupts other than [S/RESET] key may be disabled at the ANTIC chip, however.

The system events that can cause interrupts are listed below:

Chip reset - Power up

NMI - Display list interrupt (unused by OS)
Vertical blank (50/60 Hz)
[S/RESET] key

IRQ - Serial bus output ready
Serial bus output complete
Serial bus input ready
Serial bus proceed line (unused by system)
Serial bus interrupt line (unused by system)
POKEY timers 1, 2 & 4
Keyboard key
[BREAK] key
6502 BRK instruction (unused by OS)

The chip reset interrupt is vectored via location FFFC to E477 where a JMP vector to the power up routine is located. All NMI interrupts are vectored via location FFFA to the NMI interrupt service routine at E7B4 and all IRQ interrupts are vectored via location FFFE to the IRQ interrupt service routine at E6F3, at which point the cause of the interrupt must be determined by a series of tests. For some of the events there are built in monitor actions and for other events the corresponding interrupts are disabled or ignored. The system provides RAM vectors so that the user may intercept interrupts when necessary.

The remainder of section 6 will describe system actions for the various interrupt causing events, define the many RAM vectors and provide recommended procedures for dealing with interrupts.

Chip reset

Chip reset is generated in response to a power up condition. The system is completely initialized as described in section 7.

Non-maskable interrupts (NMI)

When an NMI interrupt occurs, control is transferred through the ROM vector directly to the system NMI interrupt service routine, where a cause for the interrupt is determined by examining hardware register NMIST [D40F]. If a display list interrupt is pending, a jump is made through the global RAM vector VDSLST [0200]; the OS does not use display list interrupts so VDSLST is initialized to point to an RTI instruction and must be changed by the user before a display interrupt is allowed to be generated.

If the interrupt is not a display list interrupt, then a test is made to see if it is a [S RESET] key interrupt; if so, then a jump is made to the [S RESET] initialization routine (see section 7 for details of [S RESET] initialization).

If the interrupt is neither a display list interrupt nor a [S/RESET] key interrupt then it is assumed to be a vertical blank (VBLANK) interrupt and the following actions occur:

Registers A, X & Y are pushed to the stack.

The interrupt request is cleared (NMIRES [D40F]).

A jump is made through the "immediate" vertical blank global RAM vector VVBLKI [0222] which normally points to the stage 1 VBLANK processor.

Assuming that VVBLKI has not been changed by the user, the following actions occur:

The stage 1 VBLANK processor is executed (see section 6.3.1).

Tests are made to see if a critical code section has been interrupted; if so, all registers are restored and an RTI instruction returns from the interrupt to the critical section. A critical section is determined by examining the CRITIC flag [0042] and the processor I bit; if either are set then the interrupted section is assumed to be critical.

If the interrupt was not from a critical section, then the stage 2 VBLANK processor is executed (see section 6.3.2).

A jump is then made through the "deferred" vertical blank global RAM vector VVBLKD [0224] which normally points to the VBLANK exit routine.

Assuming that VVBLKD has not been changed by the user, the following actions occur:

The 6502 A, X & Y registers are restored.

An RTI instruction is executed.

Note that there are ROM vectors to the stage 1 VBLANK processor and to the VBLANK exit routine available to the user who alters the deferred

and immediate VBLANK RAM vectors but still wants to enable normal system processes as well or restore the original vectors without having to save them. The instruction at E45F is a JMP to the stage 1 VBLANK processor; the address at [E460,2] is the value normally found in VVBLKI. The instruction at E462 is a JMP to the VBLANK exit routine; the address at [E463,2] is the value normally found in VVBLKD.

Note also that a jump is made through vector VVBLKI on every VBLANK interrupt, but a jump is made through vector VVBLKD only on interrupts from non-critical code sections.

Stage 1 VBLANK process

As part of the stage 1 VBLANK processing which will be performed at every VBLANK interrupt are the following:

The 3 byte frame counter RTCLOK [0012-0014] is incremented; RTCLOK+0 is the MSB and RTCLOK+2 is the LSB. This counter wraps to zero when it overflows (every 77 hours or so) and continues counting.

The Attract mode variables are processed as described in section 4 B10-12.

System timer 1 CDTMVI [0218,2] is decremented if it is non-zero; if the timer goes from non-zero to zero then an indirect JSR is performed via CDTMA1 [0226,2].

Stage 2 VBLANK process

As part of the stage 2 VBLANK processing which will be performed at those VBLANK interrupts which do not interrupt critical sections are the following:

The 6502 processor I bit is cleared, thus enabling the IRQ interrupts.

Various hardware registers are updated with data from the OS database as shown below.

Database item	Hardware reg.	Reason for update
SDLSTH [0231]	DLISTH [D403]	Display list end.
SDLSTL [0230]	DLISTL [D402]	
SDMCTL [022F]	DMACTL [D400]	
CHBAS [02F4]	CHBASE [D409]	
CHAOT [02F3]	CHAOTL [D401]	
GPRIOR [026F]	PRIOR [D01B]	
COLOR0 [02C4]	COLPFO [D016]	
COLOR1 [02C5]	COLPF1 [D017]	Attrace mode.
COLOR2 [02C6]	COLPF2 [D018]	

COLOR3 [02C7]	COLPF3 [D019]
COLOR4 [02C8]	COLBK [D01A]
PCOLR0 [02C0]	COLPM0 [D012]
PCOLR1 [02C1]	COLPM1 [D013]
PCOLR2 [02C2]	COLPM2 [D014]
PCOLR3 [02C3]	COLPM3 [D015]

Constant = 8 CONSOL [D01F] Console speaker off.

System timer 2 CDTMV2 [021A, 2] is decremented if it is non-zero; if the timer goes from non-zero to zero then an indirect JSR is performed through CDTMA2 [022B, 2].

System timers 3, 4 & 5 are decremented if non-zero; the corresponding flags are set to zero for each timer that changes from non-zero to zero.

Timer	Timer value	Timer flag
3	CDTMV3 [021C, 2]	CDTMF3 [022A, 1]
4	CDTMV4 [021E, 2]	CDTMF4 [022C, 1]
5	CDTMV5 [0220, 2]	CDTMF5 [022E, 1]

A character is read from the POKEY keyboard register and stored in CH [02FC] if auto-repeat is active.

The keyboard debounce counter is decremented if not equal to zero and if no key is pressed.

Keyboard auto-repeat is processed as described in section 4.5 E8.

Game controller data is read from the hardware to the RAM database as shown below.

Hardware reg.	Database item	Function
PENV [D40D]	LPENV [0235]	Lightpen.
PENH [D40C]	LPENH [0234]	
PORTA [D300]	STICK0 [0278]	Joysticks &
	STICK1 [0279]	
	PTRIGn	pot triggers.
	PTRIGO [027C]	
	PTRIG1 [027D]	
	PTRIG2 [027E]	
	PTRIG3 [027F]	
PORTB [D301]	STICK2 [027A]	
	STICK3 [027B]	
	PTRIGn	
	PTRIG4 [0280]	
	PTRIG5 [0281]	
	PTRIG6 [0282]	
	PTRIG7 [0283]	
PQTO [D200]	PADDLO [0270]	Pot controllers.

POT7 [D207]	PADDL7 [0277]	
TRIGO [D001]	STRIGO [0284]	Joystick triggers.
TRIG3 [D004]	STRIG3 [0287]	

Maskable interrupts (IRQ)

When an IRQ interrupt occurs control is transferred through the immediate IRQ global RAM vector VIMIRQ [0216]; ordinarily this vector points to the system IRQ handler whose actions are described below.

A cause for the interrupt is found by examining the IRQST [D20E] register and the PIA status registers PACTL [D302] & PBCTL [D303]. For the interrupt found, the interrupt status bit is cleared. One interrupt event is cleared and processed for each interrupt service entry; if multiple IRQs are pending a separate interrupt will be generated for each until all are serviced.

The rest of this section describes how the system IRQ interrupt service routine deals with each of the possible IRQ causing events.

The 6502 A register is pushed to the stack.

If the interrupt is due to serial I/O bus output ready, then clear the interrupt and jump through global RAM vector VSEROR [020C].

If the interrupt is due to serial I/O bus input ready, then clear the interrupt and jump through global RAM vector VSERIN [020A].

If the interrupt is due to serial I/O bus output complete, then clear the interrupt and jump through global RAM vector VSEROC [020E].

If the interrupt is due to POKEY timer #1, then clear the interrupt and jump through global RAM vector VTIMR1 [0210].

If the interrupt is due to POKEY timer #2, then clear the interrupt and jump through global RAM vector VTIMR2 [0212].

If the interrupt is due to POKEY timer #4, then clear the interrupt and fall into the following test due to a bug in the OS interrupt processor!

If the interrupt is due to a keyboard key being pressed (other than [BREAK], [START], [OPTION], [SELECT]), then clear the interrupt and jump through global RAM vector VKEYBD [0208].

If the interrupt is due to the [BREAK] key being pressed, then clear the interrupt, set the BREAK flag BRKKEY [0011] to zero, and clear the following: start/stop flag SSFLAG [02FF], cursor inhibit flag CRSINH [02F0] and Attract mode flag ATTRACT [004D]. Then return from the interrupt after restoring the 6502 A register from the stack.

If the interrupt is due to the serial I/O bus proceed line, then clear the interrupt and jump through global RAM vector VPRCED [0202].

If the interrupt is due to the serial I/O bus interrupt line, then clear the interrupt and jump through global RAM vector VINTER [0204].

If the interrupt is due to a 6502 BRK instruction, then jump through global RAM vector VBREAK [0206].

If none of the above, restore the 6502 A register and return from the interrupt (RTI).

Interrupt initialization

Whenever the system is powered up or the [S/RESET] key is pressed, the interrupt subsystem is completely re-initialized. The hardware registers are all cleared and the interrupt global RAM vectors are set to the following configurations:

Vector	Value	Type	Function
VDSLST [0200]	E7B3	NMI	RTI -- ignore interrupt.
VVBLKI [0222]	E7D1	"	System stage 1 VBLANK.
CDTMA1 [0226]	EBFO	"	SIO timeout timer.
CDTMA2 [0228]	0000	"	No system function.
VVBLKD [0224]	E93E	"	System return from int.
VIMIRQ [0216]	E6F6	IRQ	System IRQ processor.
VSEROR [020C]	EA90	"	SIO.
VSERIN [020A]	EB11	"	SIO.
VSEROC [020E]	EAD1	"	SIO.
VTIMR1 [0210]	E7B2	"	PLA, RTI -- ignore int.
VTIMR2 [0212]	E7B2	"	PLA, RTI -- ignore int.
VTIMR4 [0214]	E7B2	"	*** doesn't matter ***
VKEYBD [0208]	FFBE	"	System keyboard int. handler.
VPRCED [0202]	E7B2	"	PLA, RTI -- ignore int.
VINTER [0204]	E7B2	"	PLA, RTI -- ignore int.
VBREAK [0206]	E7B2	BRK	PLA, RTI -- ignore int.

After system initialization is complete, the interrupt enable situation is as follows:

NMI VBLANK enabled, Display list disabled.

IRQ [BREAK] key and data key interrupts enabled, all others disabled.

System timers

There are five general purpose software timers plus a frame counter supported by the OS. The timers are two bytes in length (lo,hi) and the frame counter RTCLOK [0012] is three bytes in length (hi,mid,lo). The timers count downward from any non-zero value to zero and upon reaching zero then either clear an associated flag or JSR through a RAM vector. The frame counter counts upward, wrapping to zero when it overflows. The table below shows the timers and the frame counter characteristics:

Timer name	Flag/vector	Use
* CDTMV1 [0218]	CDTMA1 [0226]	2 byte vector -- SIO timeout.
CDTMV2 [021A]	CDTMA2 [0228]	2 byte vector
CDTMV3 [021C]	CDTMF3 [022A]	1 byte flag
CDTMV4 [021E]	CDTMF4 [022C]	1 byte flag
CDTMV5 [0220]	CDTMF5 [022E]	1 byte flag
* RTCLOK [0012]		3 byte frame counter.

* These timers are maintained as part of every VBLANK interrupt (stage 1 process); the other timers are subject to the critical section test (stage 2 process) which may defer their updating to a later VBLANK interrupt.

Usage notes

This subsection describes the "tricks" that must be known in order for the user to utilize interrupts in conjunction with the operating system.

POKEY interrupt mask

ANTIC (display list & vertical blank) and PIA (interrupt & proceed lines) interrupts may be masked directly as described in the Hardware Manual. However, the POKEY interrupts ([BREAK] key, data key, serial input ready, serial output ready, serial output done and timers 1,2 & 4) are all masked by the eight bits of a single byte IRQEN [D20E] which happens to be a write-only register. Thus, in order to selectively update individual interrupt mask bits, while not changing the other bits, we must maintain a current value of that register in RAM. The name of the variable used is POKMSK [0010] and it is used as shown in the examples below:

EXAMPLE OF INTERRUPT ENABLE

```
SEI          ; TO AVOID CONFLICT WITH IRQ ...
LDA  POKMSK   ; ... PROCESSOR WHICH ALTERS VAR.
ORA  #$xx     ; ENABLE BIT(S).
STA  POKMSK
STA  IRQEN    ; TO HARDWARE REG TOO.
CLI
```

; EXAMPLE OF INTERRUPT DISABLE

```
SEI          ; TO AVOID CONFLICT WITH IRQ ...
LDA  POKMSK  ; ... PROCESSOR WHICH ALTERS VAR.
AND  #$FF-xx ; DISABLE BIT(S).
STA  POKMSK
STA  IRQEN   ; TO HARDWARE REGISTER TOO.
CLI
```

Note that the OS IRQ service routine uses and alters POKMSK, so alterations to the variable must be done with interrupts inhibited. If done at the interrupt level there is no problem, as the I bit is already set; if done at a background level then the SEI and CLI instructions should be used as shown in the examples.

Setting interrupt and timer vectors

Because vertical blank interrupts are generally kept enabled so that the frame counter RTCLOK is maintained accurately, there is a problem with setting the VBLANK vectors (VVBLKI & VVBLKD) or the timer values (CDTMV1 through CDTMV5) directly. A VBLANK interrupt could occur when only one byte of the two byte value had been updated, leading to undesired consequences. For this reason, the SETVBV [E45F] routine is provided to perform the desired update in safe manner. The calling sequence is shown below:

```
A = update item indicator
    1 - 5 for timers 1 - 5.
    6 for immediate VBLANK vector VVBLKI.
    7 for deferred VBLANK vector VVBLKD.
X = MSB of value to store.
Y = LSB of value to store.
```

JSR SETVBV

The A, X & Y registers may be altered.
The display list interrupt will always be enabled on return, even if disabled upon entry.

Note that it is possible that a vertical blank interrupt may be fully processed during a call to this routine.

When working with the system timers, the vectors for timers 1 & 2 and the flags for timers 3, 4 & 5 should be set while the associated timer is equal to zero, then the timer should be set to its (non-zero) value.

Stack content at interrupt vector points

The table below shows the stack content at every one of the RAM interrupt vector points:

VDSLST [0200]	Display list	return, P
VVBLKI [0222] *	VBLANK immed.	return, P, A, X, Y
CDTMA1 [0226]	System timer 1	return, P, A, X, Y, return
CDTMA2 [0228]	System timer 2	return, P, A, X, Y, return
VVBLKD [0224] *	VBLANK defer.	return, P, A, X, Y
VIMIRQ [0216] *	IRQ immediate	return, P
VSEROR [020C] *	Serial out rdg.	return, P
VSERIN [020A] *	Serial in rdg.	return, P
VSEROC [020E] *	Serial out cmp.	return, P
VTIMR1 [0210]	POKEY timer 1	return, P
VTIMR2 [0212]	POKEY timer 2	return, P
VTIMR4 [0214]	POKEY timer 4	return, P
VKEYBD [0208] *	Keyboard data	return, P
VPRSED [0202]	Serial proceed	return, P
VINTER [0204]	Serial interr.	return, P
VBREAK [0206]	BRK instr.	return, P

Entries flagged with '*' are initialized by the operating system at power up; changing these vectors will alter system performance if not done properly.

Miscellaneous considerations

The following paragraphs list a set of miscellaneous considerations for the writer of an interrupt service routine.

RESTRICTIONS ON CLEARING OF 'I' BIT

Display list, immediate vertical blank and system timer #1 routines should not clear the 6502 I bit. If the NMI leading to one of these routines occurred while an IRQ was being processed, then clearing the I bit will cause the IRQ to re-interrupt with unknown result.

The OS VBLANK processor carefully checks this condition after the stage 1 process and before the stage 2 process.

INTERRUPT PROCESS TIME RESTRICTIONS

If the serial I/O bus is being used, then any user defined interrupt routine plus the stage 1 VBLANK routine should not exceed 400 usec. SIO sets the CRITIC flag while serial bus I/O is in progress.

INTERRUPT DELAY DUE TO "WAIT FOR SYNC"

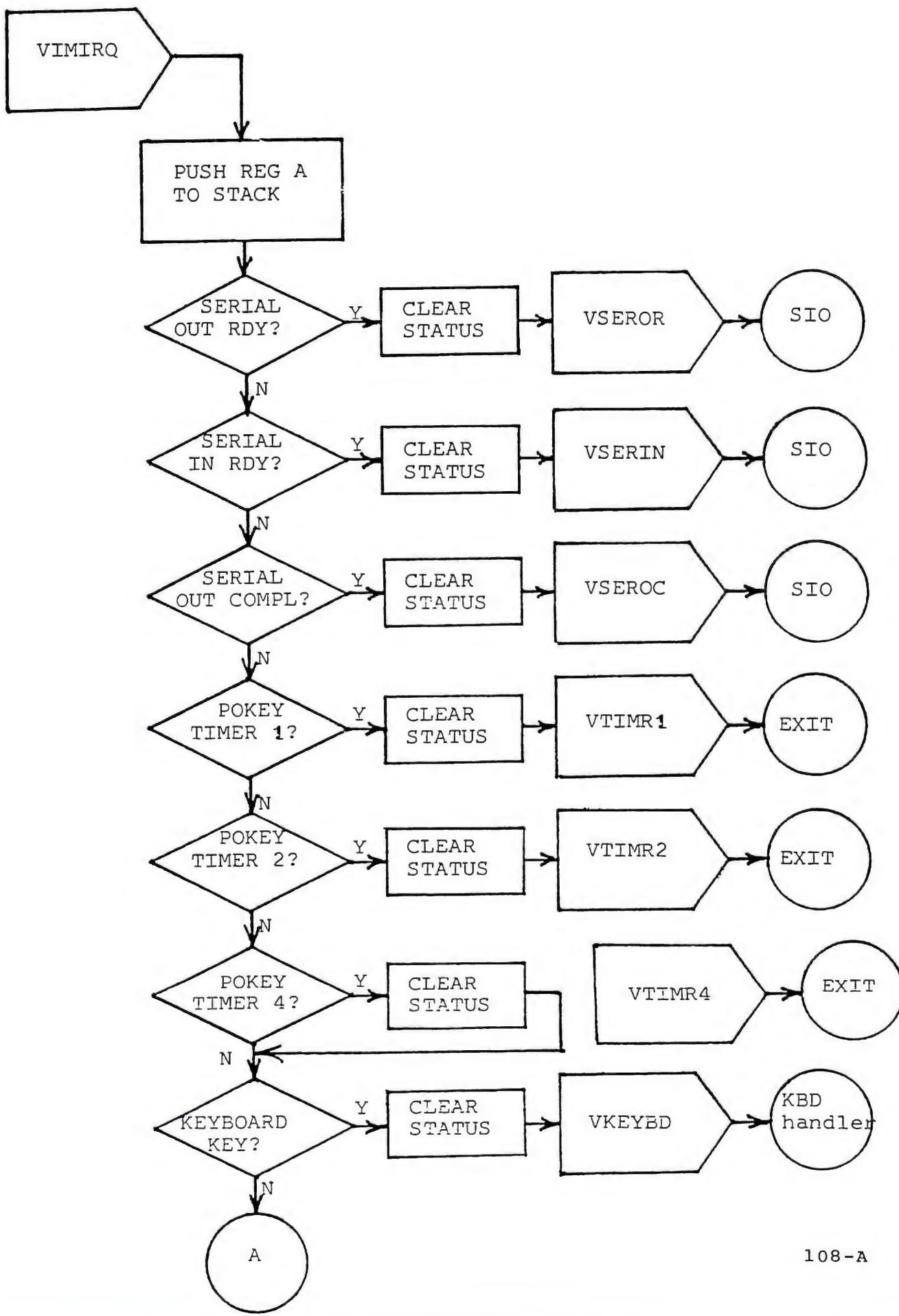
Whenever a key is read from the keyboard, the Keyboard handler sets WSYNC [D40A] repeatedly while generating the audible click on the console speaker. A problem occurs when interrupts are generated during

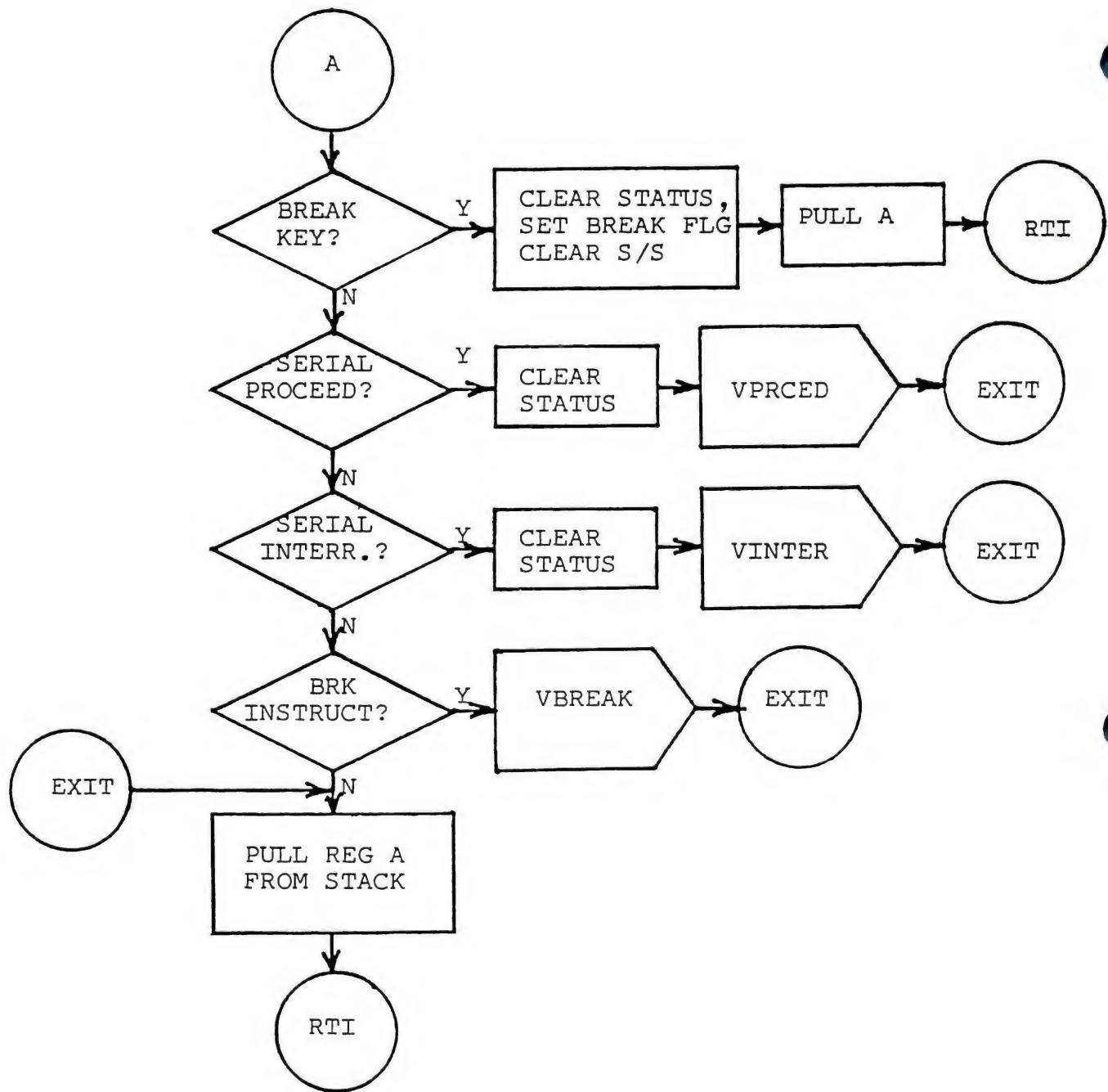
the wait for sync period; the processing of such interrupts will be delayed by one horizontal scan line. Since this condition cannot be prevented, a solution available to the user is to examine the line count VCOUNT [D4OB] and delay interrupt processing by one line when no WSYNC delay has occurred.

Flowcharts

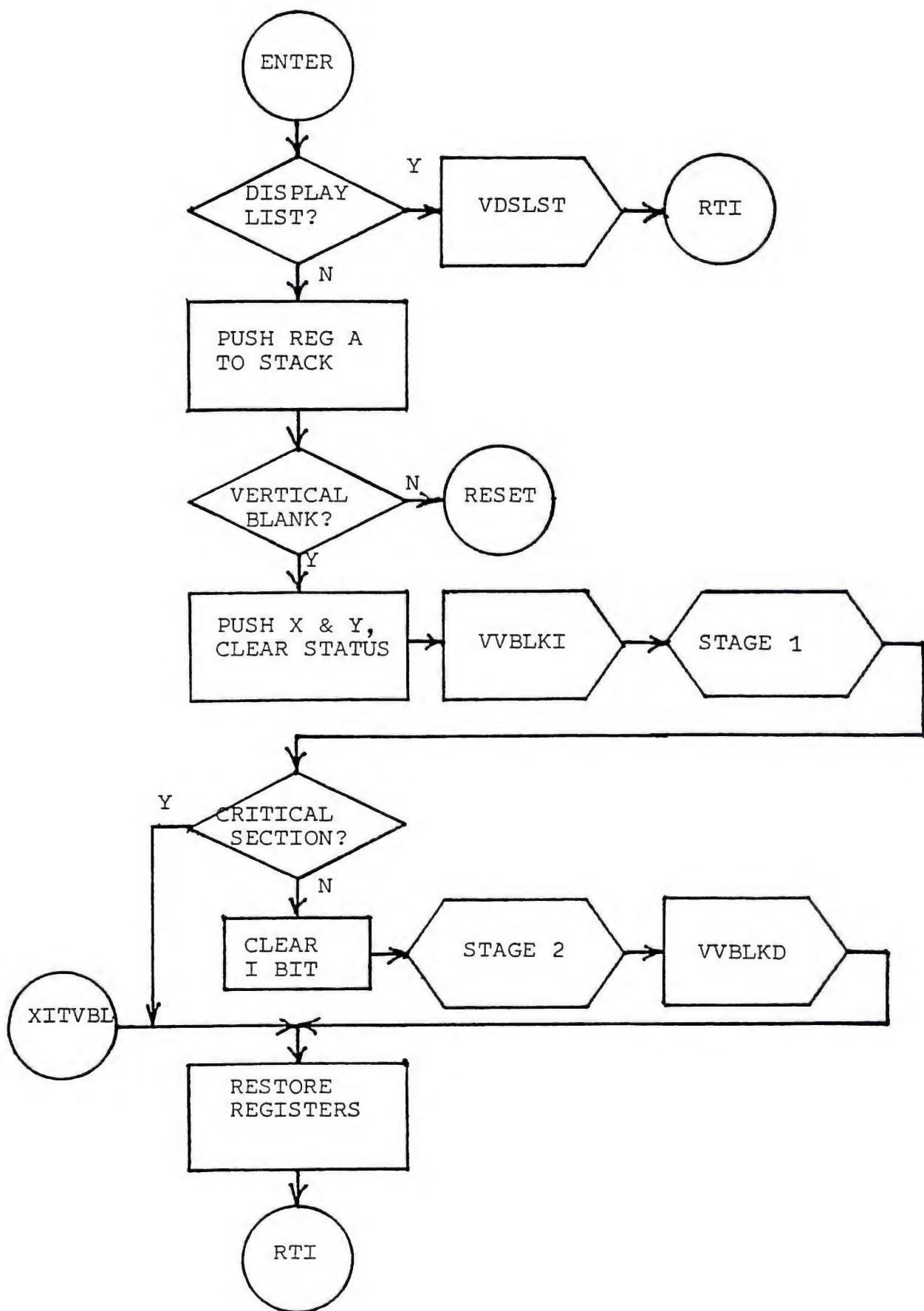
The following pages contain process flowcharts showing the main events that occur in the NMI and IRQ interrupt processes.

IRQ INTERRUPT PROCESS





NMI INTERRUPT PROCESS





7 System initialization

Introduction

System initialization takes place automatically in two circumstances: power up (also called coldstart) and the pressing of the [S/RESET] key (warmstart). In addition, there are vectors for these processes at E474 ([S/RESET]) and E477 (power up) so that they may be user initiated.

The power up initialization process is a superset of the [S/RESET] initialization process; power up initializes both the OS and user RAM regions whereas [S/RESET] initializes only the OS RAM region. In both cases the outer level software initialization entry points are called to allow the application to initialize its own variables.

Pressing the S Reset key produces an NMI interrupt and does not perform a 6502 chip S RESET. If the processor is locked up, S RESET may not be sufficient to unlock it, and the system may have to have power cycled to clear the problem.

The remainder of section 7 will discuss the details of the power up and [S/RESET] processes. Because they have many common functions (actually sharing common code), the power up process will be explained first and then the [S/RESET] process will be explained in terms of its differences from the power up process.

Power up initialization (coldstart)

The functions listed below are performed, in the order shown, as part of the power up initialization process:

1. The following 6502 processor functions are performed.
 - IRQ interrupts are disabled using the SEI instruction.
 - The decimal flag is cleared using the CLD instruction.
 - The stack pointer is set to FF.
2. The warmstart flag WARMST [0008] is set to 0 (false).
3. A test is made to see if a diagnostic cartridge is in the "A" slot:
 - Cartridge address BFFC = 00?
 - The memory at BFFC is not RAM?
 - Bit-7 of the byte at BFFD = 1?

If all of the above tests are true, then control is passed to the diagnostic cartridge via the vector at BFFE; no return is expected.

4. The lowest memory address containing non-RAM is determined by testing the first byte of every 4K "block" to see if the content can be complemented. If it can be complemented, then the original value is restored and testing continues; if it can't be

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

complemented, it is assumed to be the first non-RAM address in the system. The MSB of the address is stored temporarily in TRAMSZ [0006].

5. Zero is stored to all of the hardware register addresses shown below (most of which aren't decoded by the hardware):

D000 through D0FF
D200 through D2FF
D300 through D3FF
D400 through D4FF

6. RAM is cleared from location 0008 to the address determined in step 4 above.
7. The default value for the "non-cartridge" control vector DOSVEC [000A] is set to point to the blackboard routine. At the end of initialization, control is passed through this vector if a cartridge does not take control.
8. The coldstart flag COLDST [0244] is set to -i (local use).
9. The screen margins are set; left margin = 2, right margin = 39 for a 38 character physical line (the maximum line size of 40 characters would be obtained by setting the margins to 0 & 39). The left margin is inset because many TV sets are manufactured such that the two leftmost columns of the video picture are not entirely visible on the screen.
10. The interrupt RAM vectors VDSLST [0200] through VVBLKD [0224] are initialized; see section 6 for the initialization values.
11. Portions of the OS RAM are set to their required non-zero values as shown below:

The [BREAK] key flag BRKKEY [0011] = -i (false).

The top of memory pointer MEMTOP [02E5] = the lowest non-RAM address (from step 4); MEMTOP will be altered later when the Screen Editor is OPENed in step 15.

The bottom of memory pointer MEMLO [02E7] = 0700; MEMLO may be changed later if there is either a disk or cassette boot operation.

The following resident routines are called for initialization -- Screen Editor, Display handler, Keyboard handler, Printer handler, Cassette handler, Central I/O Monitor (CIO), Serial I/O Monitor (SIO) and the Interrupt processor.

The START key is checked, and if pressed, the cassette boot request flag CKEY [004A] is set.

12. 6502 IRQ interrupts are enabled using the CLI instruction.

13. The Device Table HATABS [031A] is initialized to point to the resident handlers. See section 9 for information relating to the device handler table.
14. The cartridge slot addresses for cartridges "B" and "A" are examined to determine if cartridges are inserted, if RAM does not extend into the cartridge address space.

If the content of location 9FFC is zero, then a JSR is executed through the vector at 9FFE, thus initializing cartridge "B". The cartridge is expected to return.

If the content of location BFFC is zero, then a JSR is executed through the vector at BFFE, thus initializing cartridge "A". The cartridge is expected to return.
15. IOCB #0 is setup for an OPEN of the Screen Editor (E) and the OPEN is performed. The Screen Editor will use the highest portion of RAM for the screen and will adjust MEMTOP accordingly. If this operation should fail, the entire initialization process is repeated.
16. A delay is effected to assure that a VBLANK interrupt has occurred. This is done so that the screen will be established before continuing.
17. If the cassette boot request flag is set (see step 11 above), then a cassette boot operation is attempted. See section 10 for details of the cassette boot operation.
18. If any of the three conditions stated below exists, an attempt is made to boot from the disk.

There are no cartridges in the slots.

Cartridge "B" is inserted and bit-0 of 9FFD is 1.

Cartridge "A" is inserted and bit-0 of BFFD is 1.

See section 10 for details of the disk boot operation.

19. The coldstart flag COLDST is reset to indicate that the coldstart process went to completion.
20. The initialization process is now complete, and the controlling application is now determined via the remaining steps.

If there is an "A" cartridge inserted and bit-2 of BFFD is 1, then a JMP is executed through the vector at BFFA.

Else, if there is a "B" cartridge inserted and bit-2 of 9FFD is 1, then a JMP is executed through the vector at 9FFA.

Else a jump is executed through the vector DOSVEC which may point to the blackboard routine (default case), cassette booted software or disk booted software. DOSVEC may be altered by the booted software as explained in section 10.

[S/RESET] initialization (warmstart)

The functions listed below are performed, in the order shown, as part of the [S/RESET] initialization process:

- A. Same as power up step 1.
- B. The warmstart flag WARMST [0008] is set to -1 (true).
- C. Same as power up steps 3 through 5.
- D. OS RAM is zeroed from locations 0200-03FF and 0010-007F.
- E. Same as power up steps 9 through 16.
- F. If a cassette boot was successfully completed during the power up initialization, then a JSR is executed through the vector CASINI [0002]. See section 10.3 for details of the cassette boot process.
- G. Same as power up step 18, except instead of booting the disk software, a JSR is executed through the vector DOSINI [000C] if the disk boot was successfully completed during the power up initialization. See section 10 for details of the disk boot process.
- H. Same as power up steps 19 and 20.

Note that the initialization procedures and main entries for all software entities are executed at every [S/RESET] as well as at power up (see steps 14, 17, 18, 20, F and G). If the user supplied initialization/startup code must behave differently in response to [S/RESET] than it does to power up, then the warmstart flag WARMST [0008] should be interrogated; WARMST = 0 means power up entry, else [S/RESET] entry.

8 Floating point arithmetic package.

This section describes the BCD floating point package that is resident in the OS ROM in both the models 400 and 800.

Introduction

The floating point package maintains numbers internally as 6 byte quantities; a 5 byte (10 BCD digit) mantissa with a 1 byte exponent. BCD internal representation was chosen so that decimal division would not lead to the rounding errors typically found in binary representations.

The package provides the following operations:

- ASCII to F.P. conversion.
- F.P. to ASCII conversion.
- Integer to F.P. conversion.
- F.P. to integer conversion.
- F.P. add, subtract, multiply and divide.
- F.P. logarithm, exponentiation and polynomial evaluation.
- F.P. zero, load, store and move.

A floating point operation is performed by calling one of the provided routines (each at a fixed address in ROM) after having set one or more floating point pseudo registers in RAM. The result of the desired operation will also involve floating point pseudo registers. The primary pseudo registers are described below and their addresses given within the square brackets:

- FRO [00D4] = 6 byte internal form of f.p. number.
- FR1 [00E0] = 6 byte internal form of f.p. number.
- FLPTR [00FC] = 2 byte pointer (lo,hi) to a f.p. number.
- INBUFF [00F3] = 2 byte pointer (lo,hi) to an ASCII text buffer
- CIX [00F2] = 1 byte index, used as offset to buffer pointed to by INBUFF [00F2].
- LBUFF [05B0] = result buffer for the FASC routine.

Functions/calling sequences

In the paragraphs that follow are the descriptions for all of the routines; unless specifically mentioned in the calling sequence, it is assumed that a pseudo register is not altered by a given routine. The numbers in square brackets [xxxx] are the ROM addresses of the routines.

ASCII to floating point conversion (Afp)

Function: This routine takes an ASCII string as input and produces a floating point number in internal form.

Calling sequence:

INBUFF = pointer to buffer containing the ASCII representation of the number.
CIX = the buffer offset to the first byte of the ASCII number.

JSR AFP [D800]
BCS first byte of ASCII number is invalid

FRO = floating point number.
CIX = the buffer offset to the first byte after the ASCII number.

Algorithm: The routine takes bytes from the buffer until it encounters a byte which cannot be part of the number. The bytes scanned to that point are then converted to a floating point number. If the first byte encountered is invalid, the carry bit is set as a flag.

Floating point to ASCII conversion (FASC)

Function: This routine converts a floating point number from internal form to its ASCII representation.

Calling sequence:

FRO = floating point number.

JSR FASC [D8E6]

INBUFF = pointer to the first byte of the ASCII number.
The last byte of the ASCII representation has the most significant bit (sign bit) set; no EOL follows.

Algorithm: The routine converts the number from its internal floating point representation to a printable form (ATASCII). The pointer INBUFF will point to part of LBUFF, where the result is stored.

Integer to floating point conversion (IFP)

Function: This routine converts a two byte unsigned integer (0 to 65535) to floating point internal representation.

Calling sequence:

FRO = integer (FRO+0 = LSB, FRO+1 = MSB).

JSR IFP [D9AA]

FRO = floating point representation of integer.

Floating point to integer conversion (FPI)

Function: This routine converts a positive floating point number from its internal representation to the nearest two byte integer.

Calling sequence:

FRO = floating point number.

JSR FPI [D9D2]

BCS f.p. number is negative or ≥ 65535.5

FRO = two byte integer ($FRO+0 = \text{LSB}$, $FRO+1 = \text{MSB}$).

Algorithm: The routine performs true rounding, not truncation, during the conversion process.

Floating point addition (FADD)

Function: This routine adds two floating point numbers and checks the result for out of range.

Calling sequence:

FRO = floating point number.

FR1 = floating point number.

JSR FADD [DA66]

BCS out of range result.

FRO = result of $FRO + FR1$.

FR1 is altered.

Floating point subtraction (FSUB)

Function: This routine subtracts two floating point numbers and checks the result for out of range.

Calling sequence:

FRO = floating point minuend.

FR1 = floating point subtrahend.

JSR FSUB [DA60]

BCS out of range result.

FRO = result of $FRO - FR1$.

FR1 is altered.

Floating point multiplication (FMUL)

Function: This routine multiplies two floating point numbers and checks the result for out of range.

Calling sequence:

FRO = floating point multiplier.
FR1 = floating point multiplicand.

JSR FMUL [DADB]
BCS out of range result.

FRO = result of FRO * FR1.
FR1 is altered.

Floating point division (FDIV)

Function: This routine divides two floating point numbers and checks for division by zero and for result out of range.

Calling sequence:

FRO = floating point dividend.
FR1 = floating point divisor.

JSR FDIV [DB28]
BCS out of range result or divisor is zero.

FRO = result of FRO / FR1.
FR1 is altered.

Floating point logarithms (LOG & LOG10)

Function: These routines take the natural or base 10 logarithms of a floating point number.

Calling sequence:

FRO = floating point number.

JSR LOG [DECD] for natural logarithm
or

JSR LOG10 [DED1] for base 10 logarithm
BCS negative number or overflow.

FRO = floating point logarithm.
FR1 is altered.

Algorithm: Both logarithms are first computed as base 10 logarithms using a 10 term polynomial approximation; the natural logarithm is computed by dividing the base 10 result by the constant LOG10(e).

The logarithm of a number Z is computed as follows:

F * (10 ** Y) = Z where 1 <= F < 10 (normalization).
L = LOG10(F) by 10 term polynomial approximation.
LOG10(Z) = Y + L.
LOG(Z) = LOG10(Z) / LOG10(e).

NOTE: This routine does not return an error if the number input is zero; the LOG10 result in this case is approximately -129.5, which is not useful.

Floating point exponentiation (EXP and EXP10)

Function: This routine exponentiates.

Calling sequence:

FRO = floating point exponent (Z).

JSR EXP [DDCO] for e ** Z

or

JSR EXP10 [DDCC] for 10 ** Z
BCS overflow.

FRO = floating point result.

FR1 is altered.

Algorithm: Both exponentials are computed internally as base 10, with the base e exponential using the identity:

e ** X = 10** (X * LOG10(e)).

The base 10 exponential is evaluated in two parts using the identity:

10 ** X = 10 ** (I + F) = (10 ** I) * (10 ** F) -- where I is the integer portion of X and F is the fraction.

The term 10 ** F is evaluated using a polynomial approximation, and 10 ** I is a straightforward modification to the floating point exponent.

Floating point polynomial evaluation (PLYEVL)

Function: This routine performs an n degree polynomial evaluation.

Calling sequence:

X, Y = pointer (X = LSB) to list of f.p. coefficients (A(i))
ordered from high order to low order.

A = number of coefficients in list.

FRO = floating point independent variable (Z).

JSR PLYEVL [DD40]

BCS overflow or other error.

FRO = result of A(n)*Z**n + A(n-1)*Z**n-1 ... + A(1)*Z +
A(0).

FR1 is altered.

Algorithm: The polynomial P(Z) = SUM(i=0 to n) (A(i)*Z**i) is computed using the standard method shown below:

$P(Z) = (\dots (A(n)*Z + A(n-1))*Z + \dots + A(1))*Z + A(0)$

Clear FRO (ZFRO)

Function: This routine sets the contents of pseudo register FRO to all zeroes.

Calling sequence:

JSR ZFRO [DA44]

FRO = zero.

Clear page zero floating point number (ZF1)

Function: This routine sets the contents of a zero page floating point number to all zeroes.

Calling sequence:

X = zero page address of f.p. number to clear.

JSR ZF1 [DA46]

zero page f.p. number(X) = zero.

Load floating point number to FRO (FLDOR and FLDOPI)

Function: These routines load pseudo register FRO with the floating point number specified by the calling sequence.

Calling sequences:

X, Y = pointer (X = LSB) to f.p. number.

JSR FLDOR [DD89]

or

FLPTR = pointer to f.p. number.

JSR FLDOPI [DD8D]

FRO = floating point number (in either case).

FLPTR = pointer to f.p. number (in either case).

Load floating point number to FR1 (FLD1R and FLDIPI)

Function: These routines load pseudo register FR1 with the floating point number specified by the calling sequence.

Calling sequences:

As in prior description, except the result goes to FR1 instead of FRO. FLD1R [DD98] and FLD1P [DD9C].

Store floating point number from FRO (FSTOR and FSTOP)

Function: These routines store the contents of pseudo register FRO to the address specified by the calling sequence:

Calling sequence:

As in prior descriptions, except the floating point number is stored from FRO rather than loaded to FRO. FSTOR [DDA7] and FSTOP [DDAB].

Move floating point number from FRO to FR1 (FMOVE)

Function: This routine moves the floating point number in FRO to pseudo register FR1.

Calling sequence:

JSR FMOVE [DDB6]

FR1 = FRO (FRO remains unchanged).

Resource utilization

The floating point package uses the following RAM locations in the course of performing the functions described in this section:

00D4 through 0OFF
057E through 05FF

If the floating package is not utilized, all of those locations are available for the user program.

Implementation details

Floating point numbers are maintained internally as 6 byte quantities, with 5 bytes (10 BCD digits) of mantissa and 1 byte of exponent. The mantissa is always normalized such that the most significant byte is non-zero (note "byte" and not "BCD digit").

The most significant bit of the exponent byte provides the sign for the mantissa, 0 for positive and 1 for negative. The remaining 7 bits of the exponent byte provide the exponent in excess 64 notation; the resulting number represents powers of 100 decimal (not powers of 10). A result of this storage format is that the mantissa holds 10 BCD digits when the value of the exponent is an even power of 10 and holds 9 BCD digits when the value of the exponent is an odd power of 10.

The implied decimal point is always to the immediate right of the first byte so that an exponent that is less than 64 indicates a number less than 1 and an exponent greater than or equal to 64 represents a number greater than or equal to 1.

Zero is represented by a zero mantissa and a zero exponent. In testing for a result from any of the standard routines, it is sufficient to test either the exponent or the first mantissa byte for zero.

The absolute value of floating point numbers must be greater than 10^{**-98} and less than 10^{**+98} or be equal to zero. There is perfect symmetry between positive and negative numbers with the exception that negative zero is never generated.

Although the precision of all computations is maintained at 9 or 10 decimal digits, the accuracy is somewhat less for those functions involving polynomial approximations (logarithm and exponentiation). Also, the problems inherent in all floating point systems are present here; for example: subtracting two very nearly equal numbers, adding numbers of disparate magnitude, or successions of any operation will all result in a loss of significant digits. For some types of applications an analysis of the data range and the order of evaluation of expressions may be required.

The remainder of this section will give some examples of the internal representation of some floating point numbers as an aid to understanding the storage format. All numbers prior to this point have been expressed in decimal notation, but the examples will use hexadecimal notation (note that 64 decimal, the excess number of the exponent, is 40 when expressed in hexadecimal).

Number: +0.02 = $2 * 10^{**-2} = 2 * 100^{**-1}$
Stored: 3F 02 00 00 00 00 (f. p. exponent = 40 - 1)

Number: -0.02 = $-2 * 10^{**-2} = -2 * 100^{**-1}$
Stored: BF 02 00 00 00 00 (f. p. exponent = 80 + 40 - 1)

Number: +37.0 = $3.7 * 10^{**1} = 37 * 100^{**0}$
Stored: 40 37 00 00 00 00 (f. p. exponent = 40 + 0)

Number: -4.60312486 * $10^{**11} = -46.03\dots * 100^{**5}$
Stored: C5 46 03 01 24 86 (f. p. exponent = 80 + 40 + 5)

Number: 0.0
Stored: 00 00 00 00 00 00 (special case)

9 - Adding new device handlers/peripherals

This section describes the interface requirements for a non resident device handler that is to be accessed via the Central I/O utility (CIO); further, the Serial bus I/O utility (SIO) interface is defined for those handlers which utilize the Serial I/O bus.

The I/O subsystem is organized with three levels of software between the user and the hardware. At the outer level is CIO, which performs the following functions:

- Logical device name to device handler mapping (on OPEN).
- I/O Control Block (IOCB) maintenance.
- Logical record handling.
- User buffer handling.

Below CIO are the individual device handlers, which perform the following functions:

- Device initialization on power up and [S/RESET].
- Device dependent support of OPEN and CLOSE commands.
- Byte at a time data input and output.
- Device dependent special operations.
- Device dependent command support.
- Device data buffer management.

At the bottom level (for Serial I/O bus peripheral handlers) is SIO, which performs the following functions:

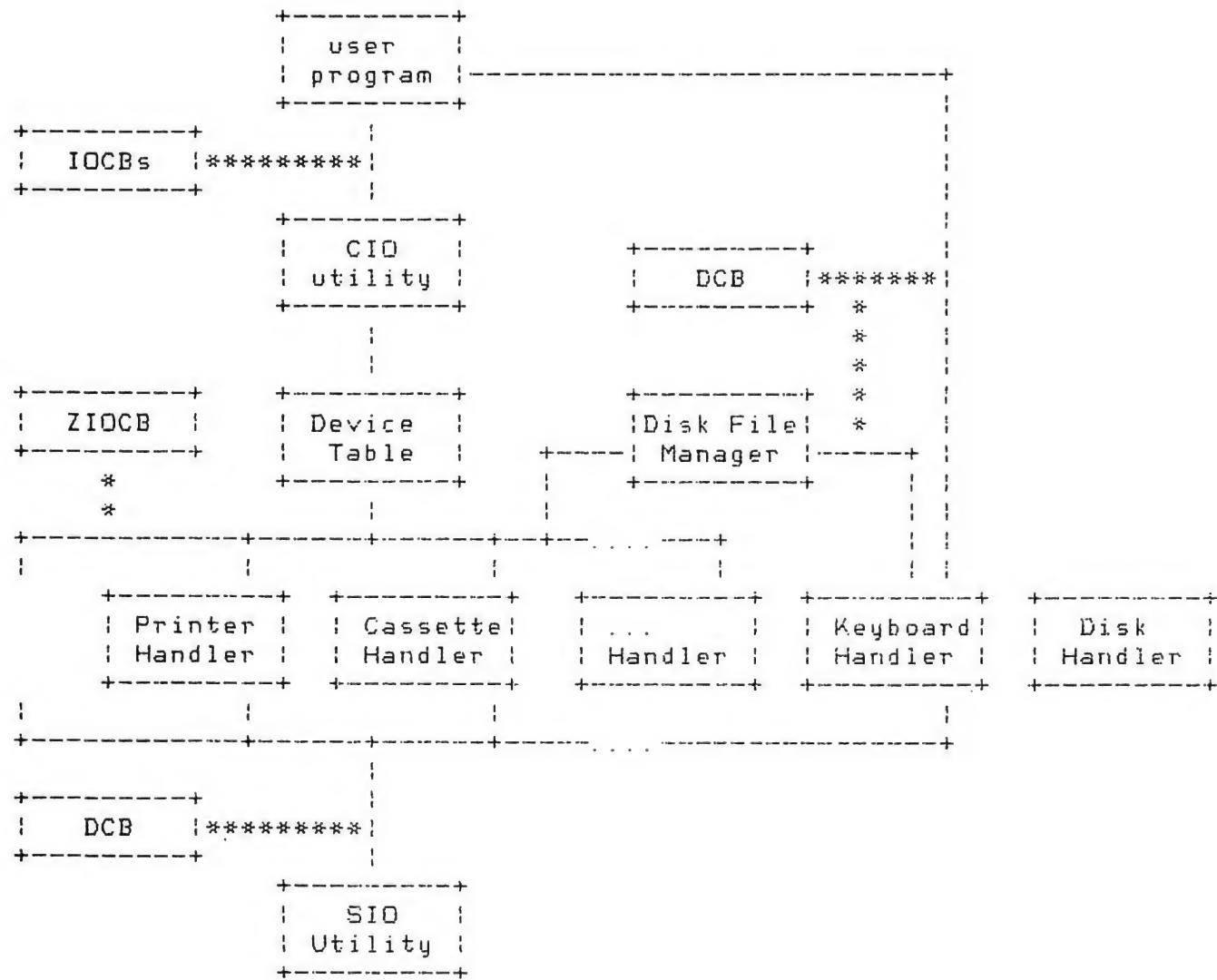
- Control of all Serial bus I/O, conforming to the bus protocol as described in section 9.
- Bus operation retries on errors.
- Return of unified error statuses on error conditions.

At each interface there is a separate control structure used for communication, as shown below:

User/CIO	I/O Control Block (IOCB)
CIO/Handler	Zero page IOCB (ZIOCB)
Handler/SIO	Device Control Block (DCB)

These relationships are shown graphically on the next page.

I/O SUBSYSTEM FLOW DIAGRAM



Where: ---- shows a control path.

**** shows the data structure required for a path.

Note the following:

1. The Keyboard/Display/Screen Editor handlers don't use SIO.
2. The Disk handler is not callable directly from CIO.
3. The DCB is shown twice in the diagram.

Device Table

The Device Table is a RAM resident table that contains the single character device name (e.g. K, D, C, etc.) and the handler address for each of the handlers known to CIO. The table is initialized at power up (and [S/RESET]) to contain entries for the following resident handlers: Keyboard (K), Display (S), Screen Editor (E), Cassette (C) and Printer (P). To install a new handler, some procedure must insert a Device Table entry after the table is initialized.

The table format is shown below:

HATABS [031A]	device name	-+
	handler vector	+-- one entry
	table address	-
	more	-+
=	entries	=
=	zero fill to	=
=	end of table	=

The table is 38 bytes long and will hold a maximum of 12 entries, with the last two bytes being zero. CIO scans the table from the end to the beginning (high to low address); so, in the case of multiple occurrences of a device name, the entry nearest the end of the table will take precedence.

The device name for each entry is a single ATASCII character, and the handler address points to the handler's vector table, which will be described in the following section.

CIO/Handler interface

This section describes the interface between the Central I/O utility and the individual device handlers that are represented in the Device Table (as described in the preceding section).

Calling mechanism

Each handler has a vector table as shown below:
space 1; need 16

OPEN vector	+ (low address)
CLOSE vector	+

```
+-----+
+ GETBYTE vector +
+-----+
+ PUTBYTE vector +
+-----+
+ GETSTAT vector +
+ SPECIAL vector +
+-----+
+ JMP init code +
+                 +      (high address)
+-----+
```

The Device Table entry for the handler points to the first byte of the vector table.

The first six entries in the table are vectors (lo,hi) which contain the address - 1 of the handler routine which handles the indicated function. The seventh entry is a 6502 JMP instruction to the handler initialization routine. CIO uses only the addresses contained in this table for handler entry; each user/CIO command translates to one or more calls to one of the handler entries defined in the vector table.

The vector table provides to CIO the handler addresses for certain fixed functions to be performed; but, in addition, operation parameters must be passed for most functions. Parameter passing is accomplished using the 6502 A, X and Y registers and an IOCB in page 0 named ZIOCB [0020]. In general, register A is used to pass data, register X contains the index to the originating IOCB and register Y is used to pass status information to CIO. The zero page IOCB is a copy of the originating IOCB; but in the course of processing some commands, CIO may alter the buffer address and buffer length parameters in ZIOCB (but not in the originating IOCB). see section 5.2.2 for information relating to the originating IOCB.

Reference Appendix B for the standard status byte values to be returned to CIO in register Y.

The following sections will describe the CIO/handler interface for each of the vectors in the handler vector table.

Handler initialization

This entry doesn't appear to have any function for non-resident handlers due to a bug in the current OS -- the Device Table is cleared in response to both [S/RESET] and power up, instead of just power up, thus preventing this entry point from ever being called. The rest of this section discusses the intended, but not implemented, use of this entry point; conformation would be in order to allow compatibility with possible corrected versions of the OS in the future.

The entry was to have been called on all occurrences of power up and [S/RESET]; the handler is to perform initialization of its hardware

and RAM data so as to assure proper processing of all CIO commands that follow.

Functions supported

This section describes the functions associated with the first six vectors from the handler vector table. A brief, device independent, description of the CIO/handler interface and recommended actions are given for each function vector.

OPEN

This entry is called in response to an OPEN command to CIO; the handler is expected to validate the OPEN parameters and perform any required device initialization associated with a device OPEN.

At handler entry, the following parameters may be of interest:

X = index to originating IOCB.

Y = \$92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).

ICBALZ/ICBAHZ [0024/0025] = address of device/filename specification.

ICAX1Z/ICAX2Z [002A/002B] = device specific information.

The handler will attempt to perform the indicated OPEN and will indicate the status of the operation by the value of the Y register. The responsibility for checking for multiple OPENS to the same device or file, where it is illegal, lies with the handler.

CLOSE

This entry is called in response to a CLOSE command to CIO; the handler is expected to release any held resources that relate specifically to that device/filename, and for output files to: 1) send any data remaining in handler buffers to the device, 2) mark the end of file, and 3) update any associated directories, allocation maps, etc.

At handler entry, the following parameters may be of interest:

X = index to originating IOCB.

Y = \$92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).

ICAX1Z/ICAX2Z [002A/002B] = device specific information.

The handler will attempt to perform the indicated CLOSE and will indicate the status of the operation by the value of the Y register.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

CIO will release the associated IOCB after the handler returns, regardless of the operation status value.

GETBYTE

This entry is called in response to a GET CHARACTERS or GET RECORD command to CIO. The handler is expected to return a single byte in the A register or return an error status in the Y register.

At handler entry, the following parameters may be of interest:

X = index to originating IOCB.

Y = \$92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).

ICAX1Z/ICAX2Z [002A/002B] = device specific information.

The handler will obtain a data byte directly from the device or from a handler maintained buffer and return to CIO with the byte in the A register and the operation status in the Y register.

Handlers which do not have short timeouts associated with the reading of data (such as the Keyboard and Cassette handlers), must monitor the [BREAK] key flag BRKKEY [0011] and return with a status of \$80 when a [BREAK] condition occurs. See section 4 E5 and section 12 for a discussion of [BREAK] key monitoring.

CIO checks for reads from device/files that have not been OPENed or OPENed for output only; the handler will not be called in those cases.

PUTBYTE

This entry is called in response to a PUT CHARACTERS or PUT RECORD command to CIO. The handler is expected to accept a single byte in the A register or return an error status in the Y register.

At handler entry, the following parameters may be of interest:

X = index to originating IOCB.

Y = \$92 (status = function not implemented by handler).

A = data byte.

ICDNOZ [0021] = device number (1-4, for multiple device handlers).

ICAX1Z/ICAX2Z [002A/002B] = device specific information.

The handler will send the data byte directly to the device or to a handler maintained buffer and return to CIO with the operation status in the Y register. If a handler maintained buffer fills, the handler will send the buffered data to the device before returning to CIO.

CIO checks for writes to device/files that have not been OPENed or OPENed for input only; the handler will not be called in those cases.

Now that the normal operation of PUTBYTE has been defined, a special case must be added; any handler that will operate within the environment of the 8K BASIC language interpreter has a different set of rules. Because BASIC can call the handler PUTBYTE entry directly, without going through CIO, the zero-page IOCB (ZIOCB) may or may not have a relation to the PUTBYTE call; thus the handler must use the outer level IOCB to obtain any information that would normally be obtained from ZIOCB. Note also in this case that the OPEN protection normally provided by CIO is bypassed (i.e. PUTBYTE to a non-OPEN device/file and PUTBYTE to a read-only OPEN).

GETSTAT

This entry is called in response to a GET STATUS command to CIO. The handler is expected to return four bytes of status to memory or return an error status in the Y register.

At handler entry, the following parameters may be of interest:

X = index to originating IOCB. Y = \$92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).
ICBALZ/ICBAHZ [0024/0025] = address of
device/filename specification.
ICAX1Z/ICAX2Z
[002A/002B] = device specific information.

The handler will get device status information from the device controller and put the status bytes in DVSTAT [02EA] through DVSTAT+3 and return to CIO with the operation status in register Y.

The IOCB need not be OPENed nor CLOSED in order for the user to request CIO to perform a GET STATUS operation; the handler must check where there are restrictions. See section 5.2.3 for a discussion of the CIO actions involved with a GET STATUS operation using both OPEN and CLOSED IOCBs, and note the impact of this on the use of the buffer address parameter.

SPECIAL

This handler entry is used to support all functions not handled by the other entry points, such as disk file RENAME, display DRAW, etc. Specifically, if the IOCB command byte value is greater than \$0D, then CIO will use the SPECIAL entry point. The handler must interrogate the command byte to determine if the requested operation is supported.

At handler entry, the following parameters may be of interest:

X = index to originating IOCB.
Y = \$92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).

ICCOMZ [0022] = command byte.
ICBALZ/ICBALH [0024/0025] = buffer address.
ICBLLZ/ICBLHZ [0028/0029] = buffer length.
ICAX1Z/ICAX2Z [002A/002B] = device specific information.

The handler will perform the indicated operation, if possible, and return to CIO with the operation status in register Y.

The IOCB need not be OPENed nor CLOSEd in order for the user to request CIO to perform a SPECIAL operation; the handler must check where there are restrictions. See section 5 for a discussion of the CIO actions involved with a SPECIAL operation using both OPEN and CLOSEd IOCBs, and note the impact of this on the use of the buffer address parameter.

Error handling

Error handling has been simplified somewhat by having CIO handle outer level errors and having SIO handle Serial bus errors, leaving the handler to process the remaining errors. These errors include:

- Out of range parameters.
- [BREAK] key abort.
- Invalid command.
- Read after end of file.

The current handlers respond to errors using the following guidelines:

- Keep the recovery simple (and therefore predictable & repeatable).

- Do not interact directly with the user for recovery instructions.

- Lose as little data as possible.

- Make all attempts to maintain the integrity of file oriented device storage -- this involves the initial design of the structural elements as well as error recovery techniques.

Resource allocation

Non-resident handlers needing code and/or data space in RAM should use the techniques listed below, in order to assure nonconflict with other parts of the OS, including other nonresident handlers.

ZERO-PAGE RAM

There are no spare bytes of zero page RAM, and even if there were, there is no allocation scheme to support multiple program assignment of the spares. Therefore, the non-resident handler must save and restore the bytes of zero-page RAM it is going to use. The bytes to use must be chosen carefully, according to the following criteria:

The bytes may not be accessed by an interrupt routine.

The bytes may not be accessed by any non-interrupt code between the time the handler modifies the bytes and then restores the original values.

A simple save/restore technique would utilize the stack in a manner similar to that shown below:

```
LDA    COLCRS      ; (for example)
PHA
LDA    COLCRS+1    ; SAVE ON STACK.
PHA

LDA    HPOINT      ; HANDLER'S POINTER.
STA    COLCRS
LDA    HPOINT+1
STA    COLCRS+1

XXX    (COLCRS),Y   ; DO YOUR POINTER THING.

PLA
STA    COLCRS+1    ; RESTORE OLD DATA.
PLA
STA    COLCRS
```

Note that for the example above, it would not be judicious to call the Display handler or the Screen Editor before restoring the original value of COLCRS, as COLCRS is a variable used by those routines.

NON-ZERO-PAGE RAM

Again, there is no allocation scheme to support the assignment of fixed regions of non-zero-page RAM to any specific process, so the handler has three choices:

1. Make a dynamic allocation at initialization time by altering MEMLO [02E7].
2. Include the variables with the handler for RAM resident handlers; this still involves altering MEMLO at the time the handler is booted.
3. If the handler is to be replacing one of the resident handlers by removing the resident handler's entry in the Device Table, then the new handler may use any RAM that the formerly resident handler would have used.

STACK SPACE

In normal situations there are no restrictions on the use of the stack by a handler; however, if the handler is planning on pushing more than a couple of dozen bytes to the stack, it should do a stack overflow test and always leave stack space for interrupt processing.

Handler/SIO interface

This section describes the interface between serial bus device handlers and the serial bus I/O utility (SIO). SIO completely handles all bus transactions following the device independent bus protocol. SIO is responsible for the following functions:

Bus data format and timing from computer end.

Error detection, retries and statuses.

Bus timeout.

Transfer of data between the bus and the caller's buffer.

Calling mechanism

SIO has a single entry point SIOV [E459] for all operations, and all parameters passed to SIO are contained in the Device Control Block (DCB) [0300], which contains the following bytes:

DEVICE BUS I. D. -- DDEVIC [0300]

The bus I. D. of the device is set by the handler prior to calling SIO (see Appendix I).

DEVICE UNIT # -- DUNIT [0301]

This byte indicates which of n units of a given device type to access and is set by the handler prior to calling SIO; in general this value comes from ICDNOZ. SIO will access the bus device whose address is equal to the value of DDEVIC plus DUNIT minus one (the lowest unit number is normally equal to one)

DEVICE COMMAND -- DCOMND [0302]

This byte is set by the handler prior to calling SIO and will be sent to the bus device as part of the command frame (see section 9 for a discussion of the command frame, and Appendix I for device command byte values).

DEVICE STATUS -- DSTATS [0303]

This byte is bi-directional; the handler will use it to indicate to SIO what to do after the command frame is sent and acknowledged, and SIO will use it to indicate to the handler the status of the requested operation.

Prior to an SIO call:

7	0
+++++-----+	
W R unused	
+-+-+-+---+-++-	

Where: W,R = 0,0 indicates no data transfer is associated with the operation.

0,1 indicates a data frame is expected from the device.

1,0 indicates a data frame is to be sent to the device.
1,1 is invalid.

After an SIO call:

7	0
+++++-----+	
status code	
+-+-+-+---+-++-	

See Appendix C for the possible SIO operation status codes.

HANDLER BUFFER ADDRESS -- DBUFLO/DBUFHI [0304/0305]

This two byte pointer is set by the handler and indicates the source or destination buffer for device data or status information.

DEVICE TIMEOUT -- DTIMLO [0306]

This byte is set by the handler and specifies the device timeout time in units of 64/60ths of a second. For example, a count of 6 specifies a timeout of 6.4 seconds.

BUFFER LENGTH/BYTE COUNT -- DBYTLO/DBYTHI [0308/0309]

This two byte count is set by the handler and indicates the number of data bytes to be transferred into or out of the buffer, for the current operation. This parameter is not required if the STATUS byte W and R bits are both zero, indicating that no data transfer is to take place.

There is a bug in SIO that causes incorrect actions when the last byte of a buffer is in a memory address ending in \$FF, such as 13FF, 42FF, etc.

AUXILIARY INFORMATION -- DAUX1/DAUX2 [030A/030B]

These two bytes are set by the handler and are included in the bus command frame by SIO; they have device specific meanings.

Functions supported

SIO does not examine the COMMAND byte it sends to the device, as all bus transactions are expected to conform to a universal protocol which includes 3 forms. These forms are stated below

(as seen from the computer):

Send command frame.

Send command frame and send data frame.

Send command frame and receive data frame.

The command form is selected purely by the values of the W and R bits in the STATUS byte as described earlier.

Error handling

SIO does the bulk of the error handling for the handler, in terms of Serial bus errors, as indicated below:

Bus timeout -- SIO provides a uniform command frame and data frame ACK byte timeout of 1/60th of a second - 0 / + 1/60th. The handler specifies the maximum COMPLETE byte timeout value in DTIMLO as described earlier.

Bus errors -- SIO detects and reports UART overrun and framing errors; the sensing of these errors in any received byte will cause the entire associated frame to be considered bad.

Data frame checksum error -- SIO validates the checksum on all received data frames and generates a checksum for all transmitted frames.

Invalid response from device -- In addition to the error conditions stated above, SIO checks that the ACK and COMPLETE responses are proper (ACK = \$41 and COMPLETE = \$43). ACK stands for acknowledge.

Bus operation retries -- SIO will attempt one complete command retry if the first attempt is not error free, where a complete command try consists of up to 14 attempts to send (and acknowledge) a command frame, followed by a single attempt to receive the COMPLETE code and possibly a data frame.

There is a bug in the retry logic for data writes, such that if the command frame is ACKed by the controller, but the data frame is not ACKed, then SIO will retry indefinitely.

Unified error status codes -- SIO provides device independent error codes as shown in Appendix C.

Serial I/O bus characteristics and protocol

This section describes the electrical characteristics of the ATARI 400 and ATARI 800 Personal Computer Systems serial bus, the use of the bus to send bytes of data, the organization of the bytes as "frames"

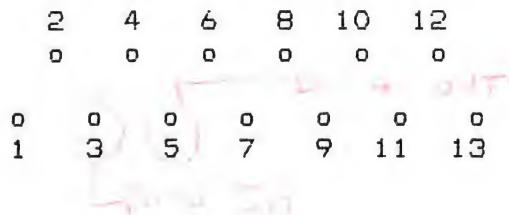
(records), and the overall command sequences which utilize frames and response bytes to provide computer/peripheral communication.

Hardware/electrical characteristics

The ATARI 400 and the ATARI 800 Personal Computer Systems communicate with peripheral devices over a 19,200 baud asynchronous serial port. The serial port consists of a serial DATA OUT (transmission) line, a serial DATA IN (receiver) line and other miscellaneous control lines.

Data is transmitted and received as 8 bits of serial data (LSB sent first) preceded by a logic zero start bit and succeeded by a logic one stop bit. The serial DATA OUT is transmitted as positive logic (+4v = one/true/high, 0v = zero/false/low). The serial DATA OUT line always assumes its new state when the serial CLOCK OUT line goes high; CLOCK OUT then goes low in the center of the DATA OUT bit time.

An end view of the Serial bus connector at the computer or peripheral is shown below (the cable connectors would of course be a mirror image):



where:
1 = computer CLOCK IN.
2 = computer CLOCK OUT.
3 = computer DATA IN.
4 = GND.
5 = computer DATA OUT.
6 = GND.
7 = COMMAND-
8 = MOTOR CONTROL.
9 = PROCEED-
10 = +5v/READY.
11 = computer AUDIO IN.
12 = +12v.
13 = INTERRUPT-

CLOCK IN is not used by the present OS and peripherals. This line can be used in future synchronous communications schemes.

CLOCK OUT is the serial bus clock. CLOCK OUT goes high at the start of each DATA OUT bit and returns to low in the middle of each bit.

DATA IN is the serial bus data line to the computer.

Pin 4 GND is the signal/shield ground line.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

DATA OUT is the serial bus data line from the computer.

Pin 6 GND is the signal/shield ground line.

COMMAND- is normally high and goes low when a command frame is being sent from the computer.

MOTOR CONTROL is the cassette motor control line (high=on, low= off).

PROCEED- is not used by the present OS and peripherals (this line is pulled high).

+5v/READY indicates that the computer is turned on and ready. This line may also be used as a +5 volt supply of 50mA current rating for ATARI peripherals only.

AUDIO IN accepts an audio signal from the cassette.

+12V is a +12 volt supply of unknown current rating for ATARI peripherals only.

INTERRUPT- is not used by the present OS and peripherals (this line is pulled high).

There are no pin reassignments made in the Serial bus cable, so pin 3, the computer's DATA IN line, is the peripheral's data output line; and similarly for pin 5.

Serial port electrical specifications

Peripheral input:

V_{IH} = 2.0v min.

V_{IL} = 0.4v max.

I_{1H} = 20ua. max. @ V_{IH} = 2.0v

I_{1L} = 5ua. max. @ V_{IL} = .4v

Peripheral output (open collector bipolar):

V_{OL} = 0.4v max. @ 1.6 ma.

V_{OH} = 4.5v min. with external 100Kohm pull-up.

Vcc/READY input.

V_{IH} = 2.0v min. @ I_{1H} = 1ma. max.

V_{IL} = 0.4v max.

Input goes to logic zero when open.

Bus commands

The bus protocol specifies that all commands must originate from the computer, and that peripherals will present data on the bus only when commanded to. Every bus operation will go to completion before another bus operation is initiated (no overlap). An error detected at any point in the command sequence will abort the entire sequence.

A bus operation consists of the following elements:

Command frame from the computer.

Acknowledgement (ACK) from the peripheral.

Optional data frame to or from the computer.

Operation complete (COMPLETE) from the peripheral.

COMMAND FRAME

The serial bus protocol provides for three types of commands: 1) data send, 2) data receive and 3) immediate (no data -- command only). There is a common element in all three types, a command frame consisting of five bytes of information sent from the computer while the COMMAND- line is held low. The format of the command frame is shown below:

-----	-----	-----	-----	-----
	device I.D.			
-----	-----	-----	-----	-----
	command			
-----	-----	-----	-----	-----
	auxilliary #1			
-----	-----	-----	-----	-----
	auxilliary #2			
-----	-----	-----	-----	-----
	checksum			
-----	-----	-----	-----	-----

The device I.D. specifies which of the serial bus devices is being addressed (see Appendix I for a list of device I.D.s).

The command byte contains a device dependent command (see Appendix I for a list of device commands).

The auxilliary bytes contain more device dependent information.

The checksum byte contains the arithmetic sum of the first four bytes (with the carry added back after every addition).

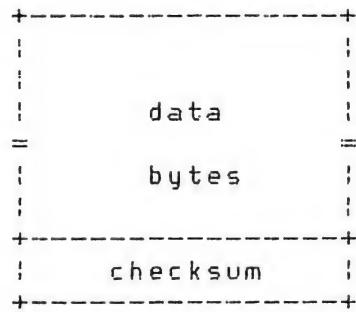
COMMAND FRAME ACKNOWLEDGE

The peripheral being addressed would normally respond to a command frame by sending an ACK byte (\$41) to the computer; if there is a

problem with the command frame, the peripheral should not respond.

DATA FRAME

Following the command frame (and ACK) may be an optional data frame which is formatted as shown below:



This data frame may originate at the computer or at the device controller, depending upon the command. Current device controllers expect fixed length data frames as does the computer, where the data frame length is a fixed function of the device type and command.

The checksum value in the data frame is the arithmetic sum of all of the frame data preceding the checksum, with the carry from each addition being added back (the same as for the command frame).

In the case of the computer sending a data frame to a peripheral, the peripheral is expected to send an ACK if the data frame is acceptable, and send a NAK (\$4E) or do nothing if the data frame is unacceptable. See the first flowchart at the end of section 9.

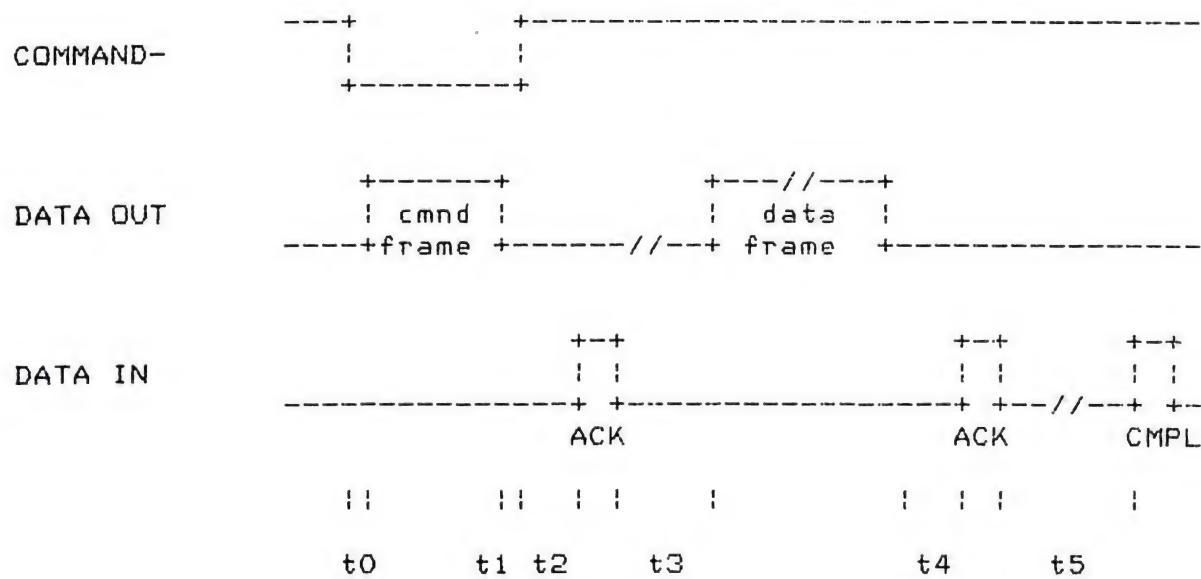
OPERATION COMPLETE

A peripheral is also expected to send an operation COMPLETE byte (\$43) at the time the commanded operation is complete. The location of this byte in the command sequence for each command type is shown in the timing diagrams which follow. If the operation cannot go to normal, error-free completion, the peripheral should respond with an ERROR byte (\$45) instead of COMPLETE.

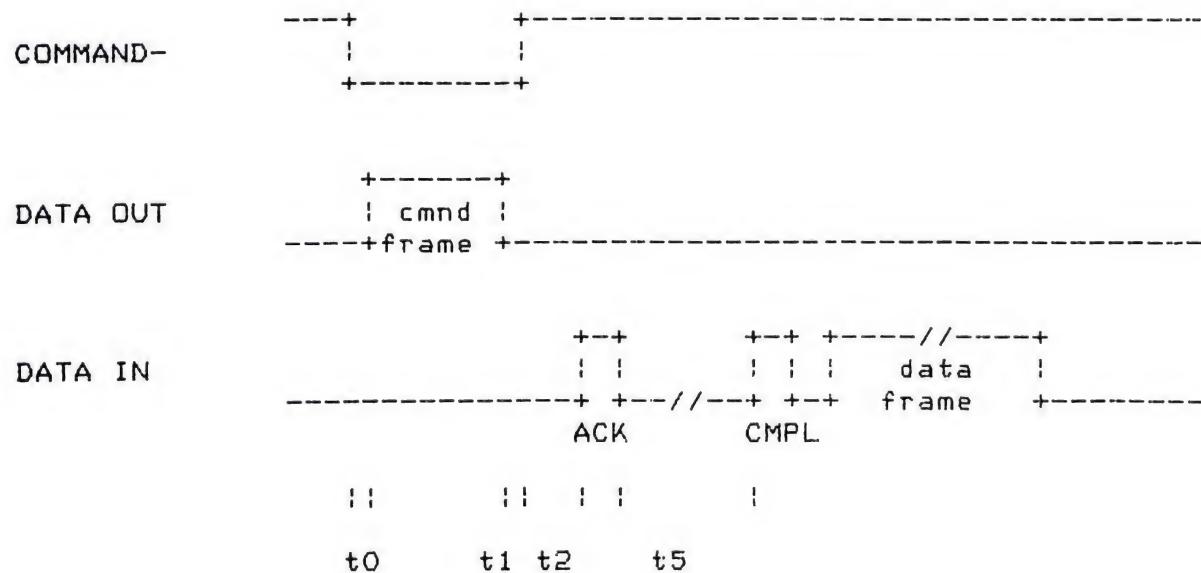
Bus timing

This section provides timing diagrams for the three types of command sequences: data send, data receive and immediate.

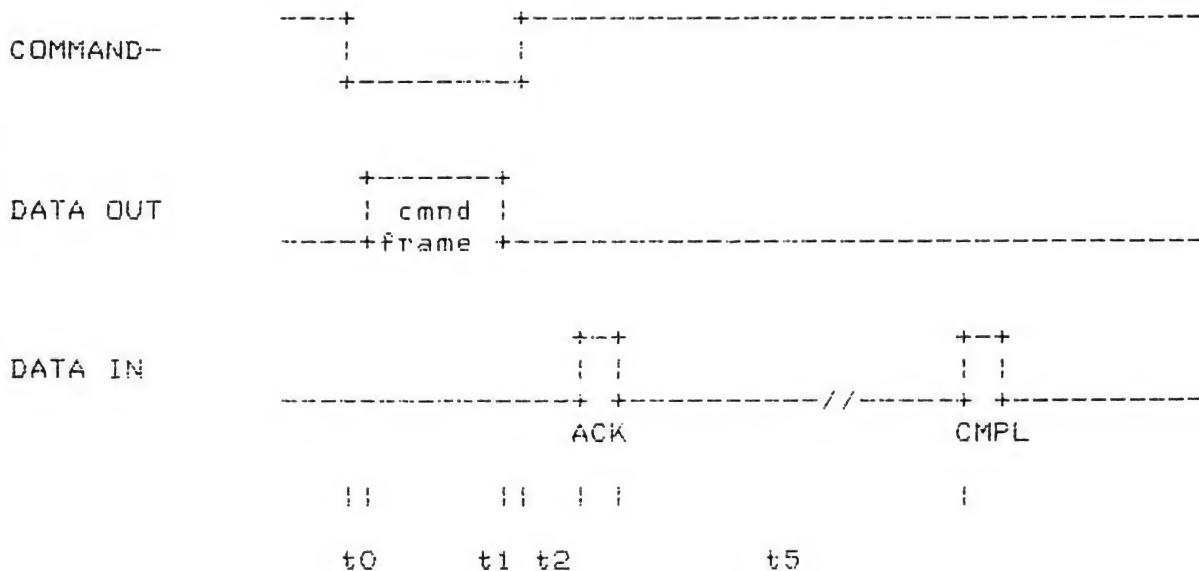
DATA SEND sequence:



DATA RECEIVE sequence:



IMMEDIATE sequence:



t_0 is the delay between the lowering of COMMAND- and the transmission of the first byte of the command frame. The computer generates this delay.

computer t_0 (min) = 750 usec.
computer t_0 (max) = 1600 usec.

t_1 is the delay between the transmission of the last bit of the command frame and the raising of the COMMAND- line. This delay is generated by the computer.

computer t_1 (min) = 650 usec.
computer t_1 (max) = 950 usec.

t_2 is the delay between the raising of COMMAND- and the transmission of the ACK byte by the peripheral. The peripheral generates this delay.

computer t_2 (min) = 0 usec.
computer t_2 (max) = 16 msec.

t3 is the delay between the receipt of the last bit of the ACK byte and the transmission of the first bit of the data frame by the computer. The computer generates this delay.

computer t3 (min) = 1000 usec.
computer t3 (max) = 1800 usec.

peripheral t3 (min) = ??
peripheral t3 (max) = ??

t4 is the delay between the transmission of the last bit of the data frame and the receipt of the first bit of the ACK byte by the computer. The peripheral generates this delay.

computer t4 (min) = 850 usec.
computer t4 (max) = 16 msec.

peripheral t4 (min) = ??
peripheral t4 (max) = ??

t5 is the delay between the the receipt of the last bit of the ACK byte and the first bit of the COMPLETE byte by the computer. The peripheral generates this delay.

computer t5 (min) = 250 usec.
computer t5 (max) = 255 sec. (handler dependent)

peripheral t5 (min) = ??
peripheral t5 (max) = N/A

Handler environment

Non-resident handlers may be installed in at least three different manners:

1. As booted software from disk or cassette.
2. Resident in a cartridge (A or B).
3. Downloaded from a serial bus device.

This section will discuss the basic mechanisms for handler installation for these environments. In order to fully utilize the information in this section, you must have read and understood the following sections:

Program environments...section 3
RAM region...section 4
Memory dynamics...section 4
System initialization...section 7
Adding new device handlers/peripherals...section 9
Program environment and initialization...section 10

9.5.1 Bootable handler

The disk or cassette booted software will want to insert the handler's vector table pointer and name to the Device Table whenever the booted software's initialization entry point is entered (on power up and [S/RESET]). Remember that both power up and [S/RESET] clear the Device Table of all but the resident handler entries.

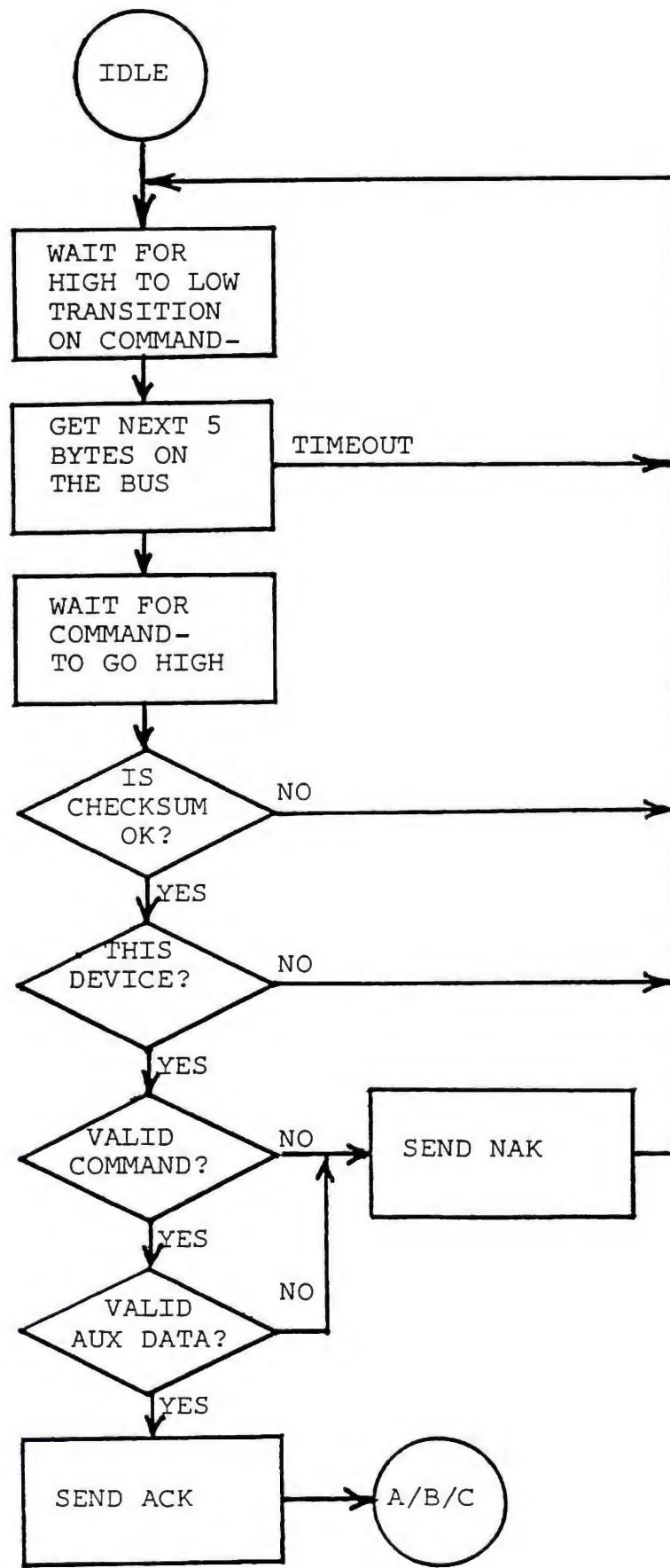
Cartridge resident handler

The cartridge software will want to insert the handler's vector table pointer and name to the Device Table whenever the cartridge's initialization entry point is entered (on power up and [S/RESET]). Remember that both power up and [S/RESET] clear the Device Table of all but the resident handler entries; therefore the device table must be reestablished by the handler initialization code upon every entry.

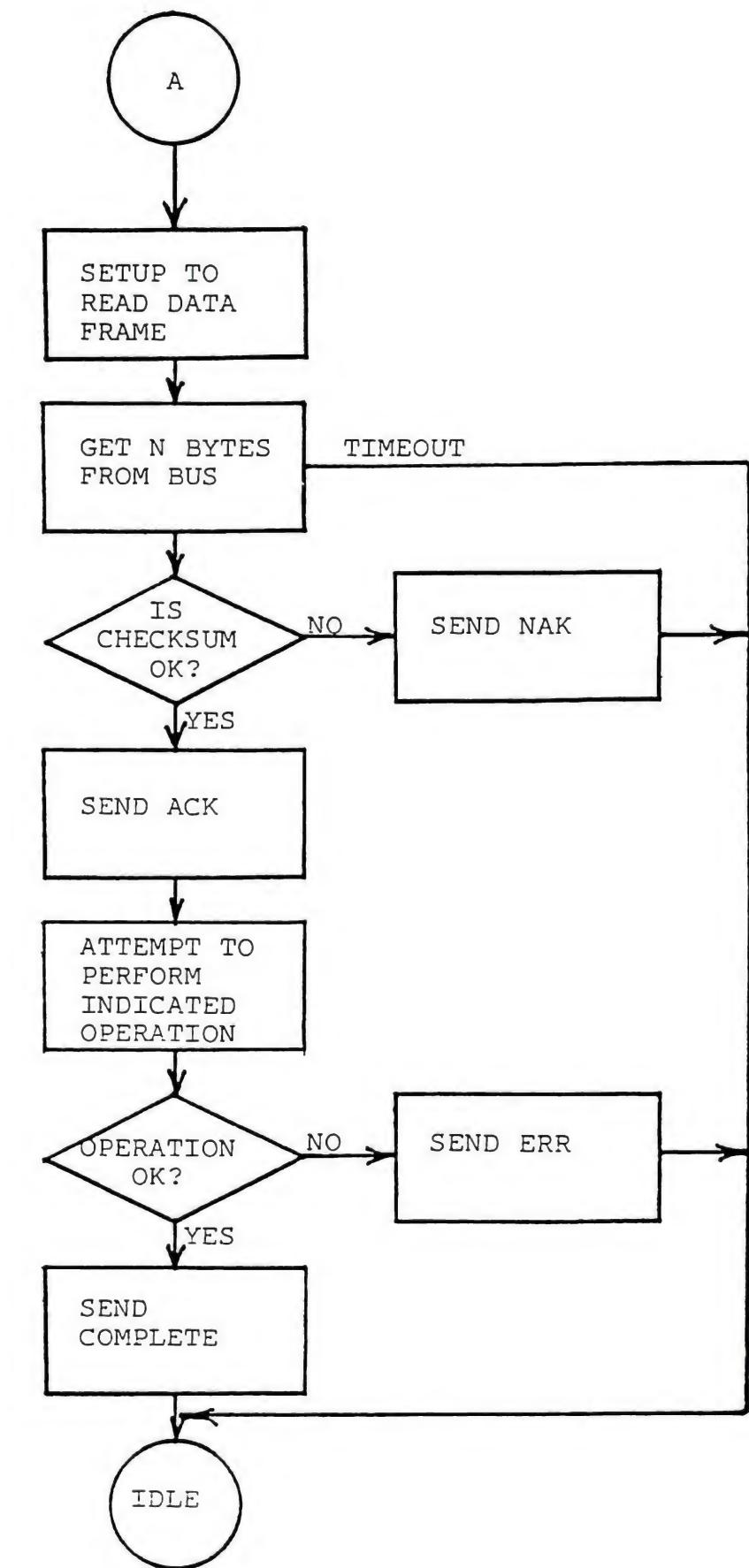
Flowcharts

The following pages contain process flowcharts showing the SIO and peripheral actions for the Serial bus command forms.

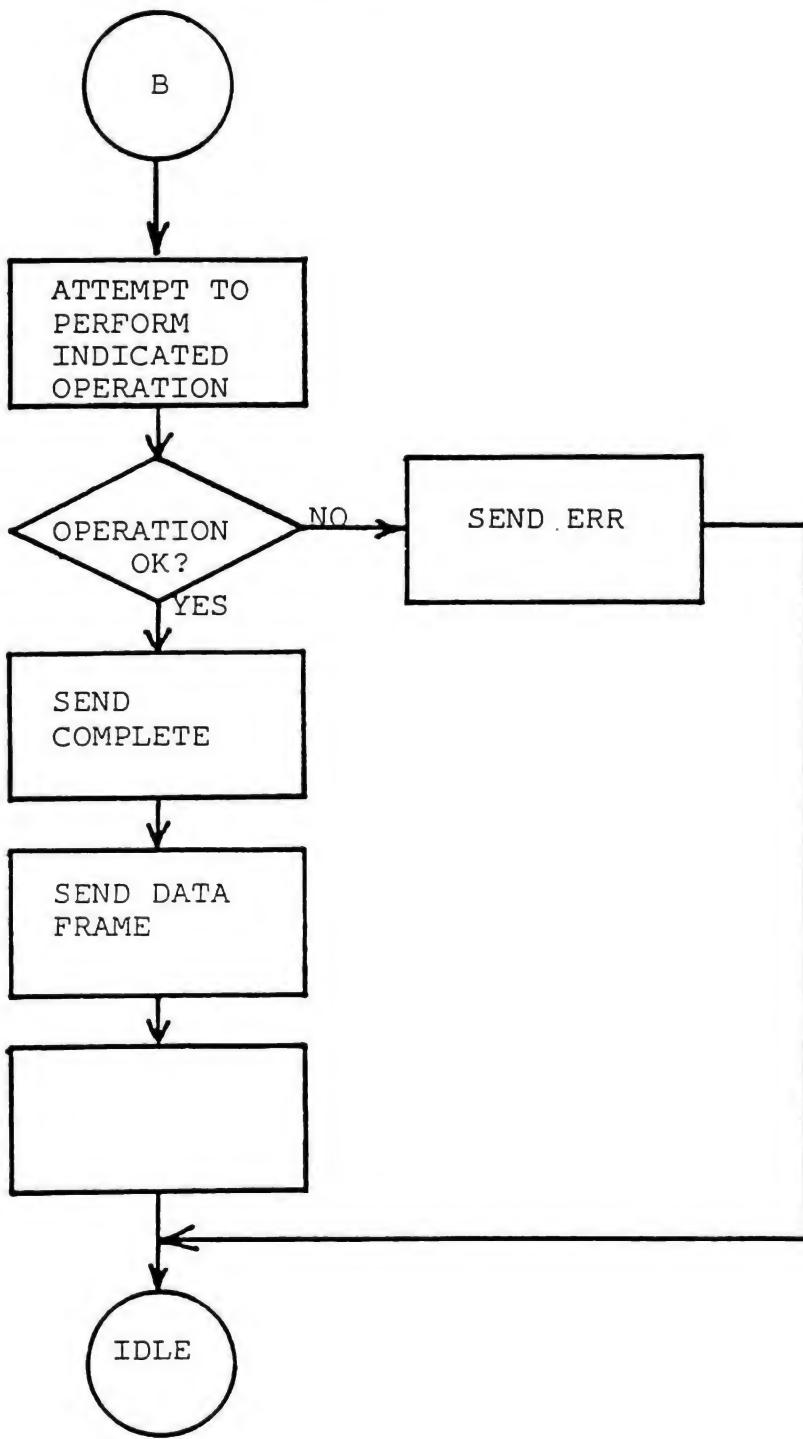
PERIPHERAL'S COMMAND FRAME PROCESSING



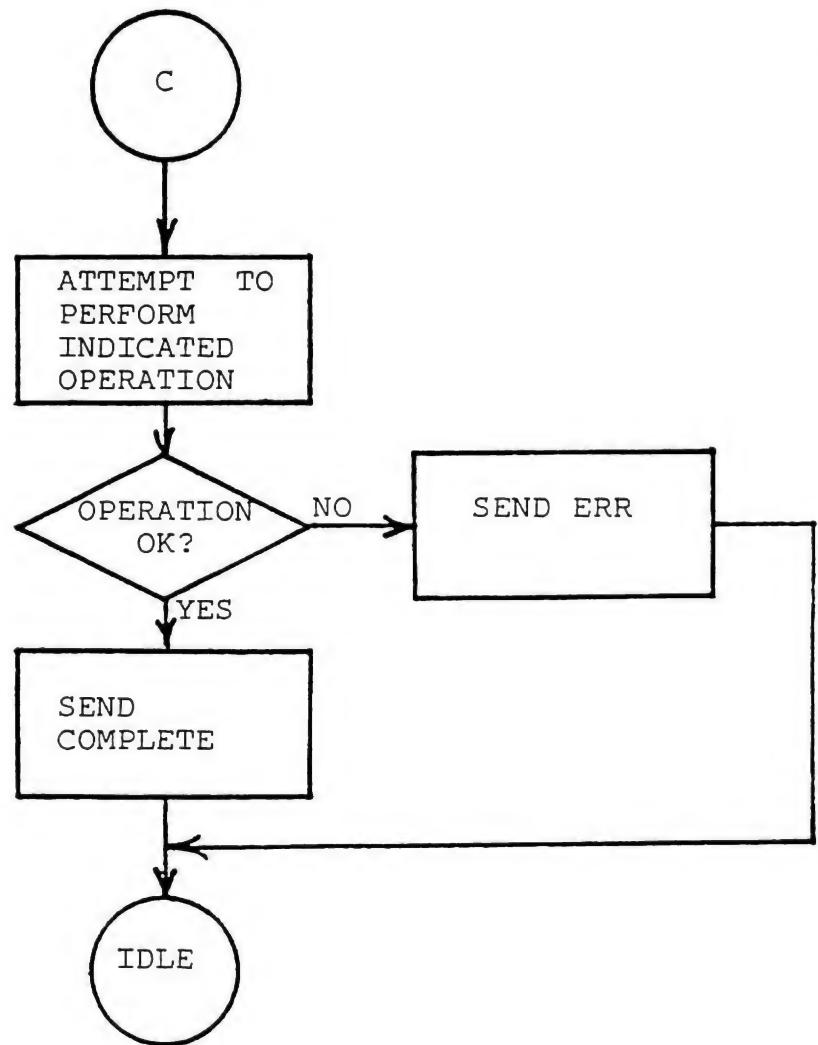
DATA FRAME TO PERIPHERAL



DATA FRAME TO COMPUTER



IMMEDIATE



10 Program environment and initialization

This section discusses several of the different software environments that are possible using the OS Configurations other than those discussed here are possible, and a thorough understanding of the power up and [S/RESET] processes (as described in section 7) will be necessary to evaluate other alternatives.

Cartridge

Most games (and some language processors) are supported via the cartridge environment. The cartridge resident software is in control of the system, sometimes using the OS and sometimes not. A cartridge can specify whether the disk is to be booted at power up time, whether the cartridge is to provide the controlling software, or whether the cartridge is a special diagnostic cartridge. These options are specified by bits in the cartridge header, as shown below:

space 1;need 13

+-----+ <td> cartridge BFFA (9FFA for cartridge B)</td>	cartridge BFFA (9FFA for cartridge B)
+-	-+
start address	
+-----+	
00	
+-----+	
option byte	
+-----+	
cartridge	
+-	-+
init address BFFF (9FFF for cartridge B)	
+-----+	

The byte of "00" is used to allow the OS to determine when a cartridge is inserted; locations BFFC and 9FFC will not read zero when there is neither RAM at those locations nor a cartridge inserted. RAM is differentiated from a cartridge by its ability to be altered.

The option byte has the following option bits:

Bit-0 = 0, then do not boot the disk.
1, then boot the disk.

Bit-2 = 0, then init but do not start the cartridge.
1, then init and start the cartridge.

Bit-7 = 0, then cartridge is not a diagnostic cartridge.
1, then cartridge is a diagnostic cartridge & control will be given to the cartridge before any of the OS is initialized (JMP (BFFE)).

The cartridge init address specifies the location to which the OS will JSR during all power up and [S/RESET] operations. As a minimum, this vector should point to an RTS instruction.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

The cartridge start address specifies the location to which the OS will JMP during all power up and [S/RESET] operations, if bit-1 of the option byte is = 1. The application should examine the variable WARMST [0008] if [S/RESET] action is to be different than power up (WARMST will be zero on power up and non-zero thereafter).

Cartridge without booted support package

A cartridge which does not specify the disk boot option and does not support the cassette boot possibility may use lower memory (from 0480 to the address in MEMTOP [02E5]) in any way it sees fit.

Cartridge with booted support package

A cartridge which does specify the disk boot option or does support the cassette boot possibility must use some care in its use of lower memory. The following regions are defined:

0480-06FF is always available to the cartridge.
MEMLO/MEMTOP region is always available to the cartridge.

Disk booted software

Software may be booted from the disk at power up time in response to one of the following conditions:

Neither Cartridge A nor B is inserted.

Cartridge A is inserted and has bit-0 of its option byte [BFFD] = 1.

Cartridge B is inserted and has bit-0 of its option byte [9FFD] = 1.

If any of these conditions are met, the OS will attempt to read the boot record from sector #1 of disk drive 1 and then transfer control to the software that was read in. The exact sequence of operations will be explained later in this section.

Disk boot file format

The key region of a disk boot file is the first 6 bytes, which are formatted as shown below:

flags	1st byte
# of sectors	
memory address	
to start load	
init	
address	6th byte
boot	
continuation	
code	

The 1st byte is stored in DFLAGS [0240], but is otherwise unused. It should equal zero.

The 2nd byte contains the number of 128 byte disk sectors to be read as part of the boot process (including the record containing this information). This number may range from 1 to 255, with 0 meaning 256.

The 3rd and 4th bytes contain the address (lo,hi) at which to start loading the first byte of the file.

The 5th and 6th bytes contain the address (lo,hi) to which the booter will transfer control after the boot process is complete and whenever the [S/RESET] key is pressed.

The Disk File Management System (FMS) has extra bytes assigned to its boot record, but this is a special case of the generalized disk boot and is discussed in section 5

Disk boot process

The disk boot process is described step by step for a configuration in which no cartridge is installed. For the general case see section 7.

1. Read the first disk record to the cassette buffer [0400].
2. Extract information from the first 6 bytes:

Save the flags byte to DFLAGS [0240, 1].
Save the # of sectors to boot to DBSECT [0241, 1].
Save the load address to BOOTAD [0242, 2].
Save the initialization address in DOSINI [000C, 2].

3. Move the record just read to the load address specified.
4. Read the remaining records directly to the load area.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

5. JSR to the load address+6 where a multi-stage boot process may continue; the carry bit will indicate the success of this operation (carry set = error, carry reset = success).
6. JSR indirectly through DOSINI for initialization of the application; the application will initialize and return.
7. JMP indirectly through DOSVEC to transfer control to the application.

Pressing the [S/RESET] key after the application is fully booted will cause steps 6 & 7 to be repeated.

Regarding step 5 -- After the initial boot process is complete, the booter will transfer control to the 7th byte of the first record; at this point the software should continue the boot process, if it is a multi-stage boot. The value of MEMLO [02E7], which should point to the first free RAM location beyond the software just booted, should be established by the booted software as shown below:

```
LDA    #END+1          ; SETUP LSB.  
STA    MEMLO  
STA    APPMHI  
LDA    #END+1/256      ; SETUP MSB.  
STA    MEMLO+1  
STA    APPMHI+1
```

If the booted software is to take control of the system at the end of the boot operation, the vector DOSVEC [000A] must be setup by the application at this time; DOSVEC points to the restart entry for the booted application. If the booted software is not to take control, then DOSVEC should remain unchanged.

```
LDA    #RESTART         ; RESTART LSB.  
STA    DOSVEC  
LDA    #RESTART/256  
STA    DOSVEC+1
```

Regarding step 6 -- The initialization point is entered on every [S/RESET] and power up; internal initialization may take place here. For controlling applications initialization may also be deferred until step 7.

Sample disk bootable program listing

Shown below is a skeletal program which can be booted from the disk and which retains control when it is entered.

```
; THIS IS THE START OF THE PROGRAM FILE.  
  
PST=  $0700          ; (OR SOME OTHER LOCATION).  
*=    PST             ; (.ORG).
```

; THIS IS THE DISK BOOT CONTROL INFORMATION.

```
.BYTE 0          ;
.BYTE PND-PST+127/128 ; NUMBER OF RECORDS.
.WORD PST         ; MEMORY ADDRESS TO START LOAD.
.WORD PINIT       ; PROGRAM INIT.
```

; THIS IS THE START OF THE BOOT CONTINUATION.

```
LDA #PND          ; ESTABLISH LOW MEMORY LIMITS.
STA MEMLO
STA APPMHI
LDA #PND/256
STA MEMLO+1
STA APPMHI+1

LDA #RESTRRT      ; ESTABLISH RESTART VECTOR.
STA DOSVEC
LDA #RESTRRT/256
STA DOSVEC+1

CLC               ; SET FLAG FOR SUCCESSFUL BOOT.
RTS
```

; APPLICATION INITIALIZATION ENTRY POINT.

```
PINIT RTS        ; NOTHING TO DO HERE FOR ...
; ... CONTROLLING APPLICATION.
```

; THE MAIN BODY OF THE PROGRAM FOLLOWS.

RESTRRT=*

; THE MAIN BODY OF THE PROGRAM ENDS HERE.

```
PND= *           ; 'PND' = NEXT FREE LOCATION.
END
```

Program and procedure to create disk boot files

This section provides a procedure that may be used to make bootable files on disks. The procedure given is not the only one possible, and no claims are made as to its elegance. The dialogue shown assumes that one is logged onto the PDP-11/34 computer from one of the development systems in the laboratories and is using LNBUG 2.0. An ATARI 400 or an ATARI 800 Personal Computer System with a disk drive is required.

User: OSL <cr>

Comp: loads the operating system.

User: DLOAD BOOTDY <cr>

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

Comp: loads the boot file maker.

User: <CTRL-P> go to LNBUG control.
X <cr>

LNBG: responds with LNBUG prompt.

User: put a formatted disk in the drive.

User: E477G <cr> starts OS and initializes mem.

LNBG: responds with LNBUG prompt in response to BRK in 'BOOTDY'.

User: R verify BRK address.

User: \$ <cr> <cr> returns to PDP-11/34 control.

Comp: PDP-11 prompt.

User: DLOAD xxxx <cr> xxxx = name of application file.

Comp: loads the application file.

User: <CTRL-P> go to LNBUG control.
X <cr>

LNBG: responds with LNBUG prompt.

User: BO01G<cr> resume 'BOOTDY'.

User: wait for completion of the disk file write.

LNBG: responds with LNBUG prompt in response to BRK in 'BOOTDY'.

User: R verify BRK address.

User: to write another boot file, type BO01G <cr>.

Shown below is a listing of the program referred to as 'BOOTDY' in the procedure above:

```
; THIS PROGRAM WRITES A SINGLE "FILE" TO THE DISK AND IS
; USED IN CONJUNCTION WITH A PROCEDURE TO MAKE DISK
; BOOTABLE FILES. THE FOLLOWING TWO SYMBOLS MUST BE EQUATED
; USING THE MEMORY LIMITS OF THE PROGRAM TO BE COPIED:
;
;      'PST' = PROGRAM START ADDRESS (SEE SAMPLE PROGRAM).
;      'PND' = PROGRAM END ADDRESS (SEE SAMPLE PROGRAM).
```

```
SECSIZ=128           ; DISK SECTOR SIZE.
PST=    $0700
PND=    $1324
FLEN=   PND-PST+SECSIZ-1/SECSIZ ; # OF SECTORS IN FILE.
```

AS/56

*= \$B000 ; THIS PROGRAM'S ORIGIN.

BOOTB BRK ; *** LOAD APPLICATION ***

; SETUP DEVICE CONTROL BLOCK FOR DISK HANDLER CALL

LDA #FLEN ; # OF SECTORS TO WRITE.
STA COUNT

LDA #1 ; DISK DRIVE #1.
STA DUNIT

LDA #'W ; SETUP FOR WRITE WITH CHECK.
STA DCOMND

LDA #PST ; POINT TO START OF APPLIC. PROG.
STA DBUFLO
LDA #PST/256
STA DBUFHI

LDA #01 ; SETUP STARTING SECTOR # = 0001.
STA DAUX1
LDA #00
STA DAUX2

; NOW WRITE THE FILE ONE SECTOR AT A TIME.

BOTO10 JSR DSKINV ; WRITE ONE SECTOR.
BMI DERR ; ERROR.

LDA DBUFLO ; INCREMENT MEMORY ADDRESS.
CLC
ADC #SECSIZ
STA DBUFLO
LDA DBUFHI
ADC #0
STA DBUFHI

INC DAUX1 ; INCREMENT SECTOR #.
BNE BOTO20
INC DAUX2

BOTO20 DEC COUNT ; MORE SECTORS TO WRITE?
BNE BOTO10 ; YES.

BRK ; STOP WHEN DONE.

DERR BRK ; STOP ON ERROR.

COUNT *=*+1 ; SECTOR COUNT.

; THIS IS THE CARTRIDGE HEADER

*= \$BFF9 ; "A" CARTRIDGE.

```
INIT    RTS
      .WORD  BOOTB
      .BYTE  0,4
      .WORD  INIT

      .END
```

Cassette booted software

Software may be booted from the cassette at power up time in much the same way as from the disk, as described in the previous section. The following requirements must be met in order to boot from the cassette:

The operator must be pressing the START key as power is applied to the system.

A cassette tape with a proper boot format file must be installed in the cassette drive, and the PLAY button must be pressed..

When the operator is given the audio prompt by the cassette handler he must press the [RETURN] key.

If all of these conditions are met, the OS will read the boot file from the cassette and then transfer control to the software that was read in. The exact sequence of operations will be explained later in this section.

Cassette boot file format

The key region of a cassette boot file is the first 6 bytes, which are formatted as shown below:

		1st byte
	# of records	
	memory address	
+-		-+
	to start load	
	init	
+-		-+
	address	6th byte

The 1st byte is not used by the cassette boot process.

The 2nd byte contains the number of 128 byte cassette records to be read as part of the boot process (including the record

containing this information). This number may range from 1 to 255, with 0 meaning 256.

The 3rd and 4th bytes contain the address (lo,hi) at which to start loading the first byte of the file.

The 5th and 6th bytes contain the address (lo,hi) to which the booter will transfer control after the boot process is complete and whenever the [S/RESET] key is pressed.

Cassette boot process

The cassette boot process is described step by step for a configuration in which no cartridge is installed and no disks are attached. For the general case see Section 7.

1. Read the first cassette record to the cassette buffer.
2. Extract information from the first 6 bytes:
 - Save the # of records to boot.
 - Save the load address.
 - Save the initialization address in CASINI [0002].
3. Move the record just read to the load address specified.
4. Read the remaining records directly to the load area.
5. JSR to the load address+6 where a multi-stage boot process may continue; the carry bit will indicate the success of this operation (carry set = error, carry reset = success).
6. JSR indirectly through CASINI for initialization of the application; the application will initialize and return.
7. JMP indirectly through DOSVEC to transfer control to the application.

Pressing the [S/RESET] key after the application is fully booted will cause steps 6 & 7 to be repeated.

Regarding step 5 -- After the initial boot process is complete, the booter will transfer control to the 7th byte of the first record; at this point the software should continue the boot process (if it is a multi-stage boot) and then stop the cassette drive, which due to a system bug will still be running, using the following instruction sequence:

```
LDA    #$3C
STA    PACTL [D302]
```

The application should then set a value in MEMLD [02E7] which points to the first free RAM location beyond the software just booted, as

shown below:

```
LDA    #END+1      ; SETUP LSB.  
STA    MEMLO  
STA    APPMHI  
LDA    #END+1/256   ; SETUP MSB.  
STA    MEMLO+1  
STA    APPMHI+1
```

If the booted software is to take control of the system at the end of the boot operation, the vector DOSVEC [000A] must be setup by the application at this time; DOSVEC points to the restart entry for the booted application. If the booted software is not to take control, then DOSVEC should remain unchanged.

```
LDA    #RESTRT     ; RESTART LSB.  
STA    DOSVEC  
LDA    #RESTRT/256  
STA    DOSVEC+1
```

Regarding step 6 -- The initialization point is entered on every [S/RESET] and power up; internal initialization may take place here. For controlling applications initialization may also be deferred until step 7.

Sample cassette bootable program listing

Shown below is a skeletal program which can be booted from the cassette and which retains control when it is entered.

; THIS IS THE START OF THE PROGRAM FILE.

```
PST= $0700          ; (OR SOME OTHER LOCATION).  
*= PST             ; (.ORG).
```

; THIS IS THE CASSETTE BOOT CONTROL INFORMATION.

```
.BYTE 0           ; (DOESN'T MATTER).  
.BYTE PND-PST+127/128 ; NUMBER OF RECORDS.  
.WORD PST         ; MEMORY ADDRESS TO START LOAD.  
.WORD PINIT       ; PROGRAM INIT.
```

; THIS IS THE START OF THE BOOT CONTINUATION.

```
LDA    #$3C          ; STOP THE CASSETTE.  
STA    PACTL
```

```
LDA    #PND          ; ESTABLISH LOW MEMORY LIMITS.  
STA    MEMLO  
STA    APPMHI  
LDA    #PND/256
```

```
STA      MEMLO+1
STA      APPMHI+1

LDA      #RESTRT          ; ESTABLISH RESTART VECTOR.
STA      DOSVEC
LDA      #RESTRT/256
STA      DOSVEC+1

CLC          ; SET FLAG FOR SUCCESSFUL BOOT.
RTS

; APPLICATION INITIALIZATION ENTRY POINT.

PINIT RTS           ; NOTHING TO DO HERE FOR ...
; ... CONTROLLING APPLICATION.

; THE MAIN BODY OF THE PROGRAM FOLLOWS.

RESTRT=*
; THE MAIN BODY OF THE PROGRAM ENDS HERE.

PND=    *           ; 'PND' = NEXT FREE LOCATION.
END
```

Program and procedure to create cassette boot files

This section provides a procedure and a program listing that may be used to make bootable files on cassette tapes. The procedure given is not the only one possible, and no claims are made as to its elegance. The dialogue shown assumes that one is logged onto the PDP-11/34 computer from one of the development systems in the laboratories and is using LNBUG 2.0. An ATARI 400 or ATARI 800 Personal Computer System with an ATARI 410 Program Recorder is also required.

User: DSL <cr>

Comp: loads the operating system.

User: DLOAD BOOTCY <cr>

Comp: loads the boot file maker.

User: <CTRL-P> go to LNBUG control.
X <cr>

LNBG: responds with LNBUG prompt.

User: E477G <cr> starts OS and initializes mem.

User: wait for tone indicating cassette write request.

User: <CTRL-C> interrupts the initialized prog.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

LNBG: responds with LNBUG prompt.

User: \$ <cr> <cr> returns to PDP-11/34 control.

Comp: PDP-11 prompt.

User: DLOAD xxxx <cr> xxxx = name of application file.

Comp: loads the application file.

User: <CTRL-P> go to LNBUG control.
X <cr>

LNBG: responds with LNBUG prompt.

User: P resume 'BOOTCY'.

User: setup cassette drive to record on tape.

User: press the [RETURN] key on the Model 400/800 keyboard.

User: wait for completion of the cassette file write.

LNBG: responds with LNBUG prompt in response to BRK in 'BOOTCY'.

User: R verify BRK address.

User: to write another boot file, type BOOOG <cr>.

Shown below is a listing of the program referred to as 'BOOTCY' in the procedure above:

```
; THIS PROGRAM WRITES A SINGLE FILE TO THE CASSETTE AND IS
; USED IN CONJUNCTION WITH A PROCEDURE TO MAKE CASSETTE
; BOOTABLE FILES. THE FOLLOWING TWO SYMBOLS MUST BE EQUATED
; USING THE MEMORY LIMITS OF THE PROGRAM TO BE COPIED:
;
; 'PST' = PROGRAM START ADDRESS (SEE SAMPLE PROGRAM).
; 'PND' = PROGRAM END ADDRESS (SEE SAMPLE PROGRAM).
```

```
PST= $0700
PND= $1324
FLEN= PND-PST+127/128*128 ; ROUND UP TO MULTIPLE OF 128.

*= $B000 ; THIS PROGRAM'S ORIGIN.

BOOTB LDX #\$10 ; USE IOCB #1.

; FIRST OPEN THE CASSETTE FILE FOR WRITING.

LDA #OPEN ; SETUP FOR DEVICE "OPEN".
STA ICCOM, X
```

```
LDA    #OPNOT           ; DIRECTION IS "OUTPUT".
STA    ICAX1, X
LDA    #$80             ; SELECT SHORT IRG.
STA    ICAX2, X

LDA    #CFILE            ; SETUP POINTER TO DEVICE NAME.
STA    ICBAL, X
LDA    #CFILE/256
STA    ICBAH, X

JSR    CIOV              ; ATTEMPT TO OPEN FILE.
BMI    CERR              ; ERROR.

; NOW WRITE THE ENTIRE FILE AS ONE OPERATION.

LDA    #PUTCHR           ; SETUP FOR "PUT CHARACTERS".
STA    ICCOM, X

LDA    #PST               ; POINT TO START OF APPLIC. PROG.
STA    ICBAL, X
LDA    #PST/256
STA    ICBAH, X

LDA    #FLEN              ; SETUP # OF BYTES TO WRITE.
STA    ICBLL, X
LDA    #FLEN/256
STA    ICBLH, X

JSR    CIOV              ; WRITE ENTIRE FILE.
BMI    CERR              ; ERROR.

; NOW CLOSE THE FILE AFTER SUCCESSFUL WRITE.

LDA    #CLOSE             ; SETUP FOR "CLOSE".
STA    ICCOM, X

JSR    CIOV              ; CLOSE THE FILE.
BMI    CERR              ; ERROR.

BRK                ; STOP WHEN DONE.

CERR   BRK              ; STOP ON ERROR.

CFILE  BYTE   "C:", CR      ; FILE NAME.

; THIS IS THE CARTRIDGE HEADER

*=    $BFF9              ; "A" CARTRIDGE.

INIT   RTS
      .WORD  BOOTB
      .BYTE  0, 4
      .WORD  INIT
```

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT APPROVAL
OPERATING SYSTEM, CO16555

END

11. Advanced techniques and application notes

This section presents information which may be of use to the user who wishes to use the capabilities of the OS and use some of the hardware capabilities that aren't directly available through the OS and, in fact, may be in direct conflict with parts of the OS.

Sound generation

The OS uses the POKEY sound generation capabilities only in the I/O subsystem, for cassette FSK tone generation and for the "noisy bus" option in SIO.

Capabilities

The hardware provides 4 independently programmable audio channels which are mixed and sent to the television set as part of the composite video signal. The POKEY registers shown below are all concerned with sound control (as described in the ATARI Personal Computer System HARDWARE MANUAL).

AUDCTL [D208]	Audio control.
AUDC1 [D201] & AUDF1 [D200]	Channel 1 control.
AUDC2 [D203] & AUDF2 [D202]	Channel 2 control.
AUDC3 [D205] & AUDF3 [D204]	Channel 3 control.
AUDC4 [D207] & AUDF4 [D206]	Channel 4 control.

Conflicts with OS

There are two potential conflicts with the OS involving sound generation:

The OS may generate its own sounds and then turn off all sounds as part of I/O operations to the cassette and the serial bus peripherals.

The OS does not turn off sounds on [S/RESET] or [BREAK]; if the sounds are to be turned off under those conditions, the controlling program must provide that capability.

Screen graphics

Hardware capabilities

The hardware capabilities for screen presentations are quite versatile; the OS uses a very small amount of the capability provided. The means of extension, however, are non-trivial; and making changes to a screen format while still utilizing the resident Display handler

will be difficult. See the ATARI Personal Computer System HARDWARE MANUAL for information regarding screen presentations.

OS capabilities

The resident Display handler arbitrarily supports 8 of the 11 possible full screen modes (11 of 14 modes if the GTIA chip is used in place of the CTIA), and allows for an optional "split screen" text window of fixed size. The hardware allows for many more options than the Display handler supports, as will be seen by reading the ATARI Personal Computer System HARDWARE MANUAL.

Cursor control

The Display handler text and graphics cursors may be directly controlled by the user as described in section 5 and in Appendix K. B1-4.

Color control

The color register assignments that the Display handler makes upon all OPEN commands may be altered by the user as described in Appendix K and elsewhere. Note that every [S/RESET] or Display handler OPEN will reset the values back to the system default.

Alternate Character Sets

In screen text modes 1 and 2, two character sets are available, the sets being selectable by the value stored in database variable CHBAS [02F4]/ tthe default value of \$E0 provides capital (upper case) letters, numbers and the punctuation characters)corresponding to display codes \$20 through \$5F in Appendix E); the alternate value of \$E2 provides lower case letters and the special character graphics set (corresponding to display codes \$60 through \$7F and \$OO through \$1F in appendix E).

In addition, user defined character sets may be obtained for text modes 0, 1 and 2 by provided the character matrix definitions in RAM and setting CHBAS to point to those definitions. CHBAS always contains the most significant bits of the memory address of the start of the character definitions, as shown below:

	7	0
	+---+---+---+---+	
CHBAS		Test mode 0
	+---+---+---+---+	
	+---+---+---+---+	
	+---+---+---+---+	

Where X indicates an ignored address bit (assumed = 0).

Each character is defined by an 8 x 8 bit matrix; the character
' is defined as shown below:

7	0 Byte
0	
1	
2	
3	
4	
5	
6	
7	

The storage for the character set involves 8 consecutive bytes for each character with characters ordered consecutively by their internal code value (see the discussion in appendix K relating to B55).

Character base	Character for	
	Code \$00	8 bytes
	Character for	
	Code \$0/	increasing addresses
	Character for	
	Code \$7E	
	Character for	
	Code \$7F	

Players/missiles

The OS makes no use of the player/missile generation capability of the hardware; but, luckily, it may be used independently of the OS with no conflict.

Hardware capabilities

The hardware allows a number of independently moveable screen objects (of limited width) to be positioned and moved about the screen without affecting the "playfield" (bit mapped graphics or character) data. Priority control allows the various objects to have a display precedence in case of conflict (overlap).

Conflicts with OS

The only potential problem is that the user must assure that the player/missile data is address aligned as required by PMBASE [D407]; and finding a suitable free area that is guaranteed to be free under all environments could be a problem.

Reading game controllers

The game controllers shown below are read by the OS as part of the stage 2 VBLANK process (see Appendix K J1-9):

Joysticks/triggers 1-4.
Paddle controllers/triggers 1-8.
Driving controllers/triggers 1-4.
Lightpen/trigger.

In addition to these controllers, other information may be sensed or sent using the PIA chip to which the console connectors are interfaced.

Keyboard controller sensing

The PASCAL procedure below shows how to read data from an ATARI keyboard controller which is connected to the first port; the hardware register and OS database names used are from the OS equate file.

```
FUNCTION READKEY (DRIVEVAL:BYTE):BYTE;
BEGIN
  PORTA := DRIVEVAL { set row select };
  DELAY           { wait for OS to read data };
  KCODE := 0       { preset for no key read };
  IF PADDLx > 10 THEN KCODE := 1 { column 1 };
  IF PADDLx > 10 THEN KCODE := 2 { column 2 };
  IF STRIGx = 0  THEN KCODE := 3 { column 3 };
  READKEY := KCODE { set function value }
END;

BEGIN

{ setup PIA port A for 4 bits out };110010/10025setup PIA
port A for 8 bits out
PACTL := $30      { direction register select };
PORTA := $FF       { set direction bits for output };
PACTL := $34       { data register select };

{ setup of driving values, each selects a different row }
DVAL[0] := $EE;
DVAL[1] := $DD;
DVAL[2] := $BB;
DVAL[3] := $77;

REPEAT { loop to read the controller keys }

I := 0;
REPEAT
  KEY := READKEY (DVAL[I]) { read a row };
  IF KEY <> 0 THEN KEY := KEY + (I*3) { encode };
  I := I + 1             { next row }
UNTIL (I > 3) OR (KEY <> 0);
IF KEY <> 0 THEN WRITELN ('KEY VALUE = ',KEY)
  UNTIL FALSE { forever! }

END.
```

The table below shows the variable/register values used for reading a keyboard controller from each of the four controller ports.

Port 1	Port 2	Port 3	Port 4
+++	+++	+++	+++

```

| Port A | OF   | FO   |           |
| direction |       |       |           |
| bits    |       |       |           |
+-----+-----+-----+-----+-----+
| Port B | -   | -   | OF   | f FO |
| direction |       |       |           |
| bits    |       |       |           |
+-----+-----+-----+-----+-----+
| Port A | FE, FD, IEF, DF |           | | |
| row sel | FB, F7 | BF, 7F | -   | -   |
| lect   |       |       |       |       |
+-----+-----+-----+-----+-----+
| Port B |           | IFE, FD, FB, IREF, DF, | |
| row sel |           | F7           | BF, 7F |
| lect   |           |           |       |
+-----+-----+-----+-----+-----+
| Column 1|PADDL 1|PADDL 3|PADDL 5 |PADDL 7 |
| Sense   |       |       |       |       |
+-----+-----+-----+-----+-----+
| Column 2|PADDL 0|PADDL 2|PADDL 4 |PADDL 6 |
| Sense   |       |       |       |       |
+-----+-----+-----+-----+-----+
| Column 3|STRIG 0|STRIG 1|STRIG 2 |STRIG 3 |
| Sense   |       |       |       |       |
+-----+-----+-----+-----+-----+

```

Front panel connectors as I/O ports

The three pages that follow show how some of the pins in the front panel (game controller) connectors can be used as general I/O pins.

OO

ATARI 400/800 Front Panel (Controller) Jack as I/O Ports

Hardware Information:

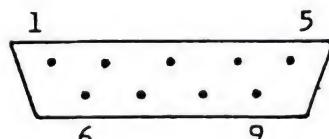
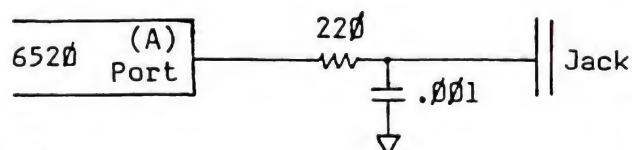
PIA (6520 / 6820)

Out: TTL levels, 1 load

In : TTL levels, 1 load

For more information refer to 6520 chip manual.

Port A Circuit (typical):

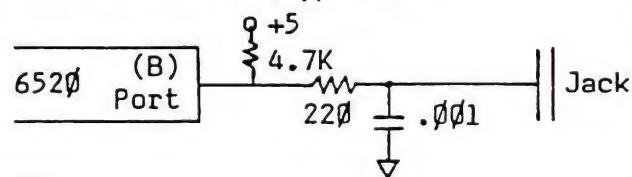


Male connector, FRONT view

Pin 8= Ground

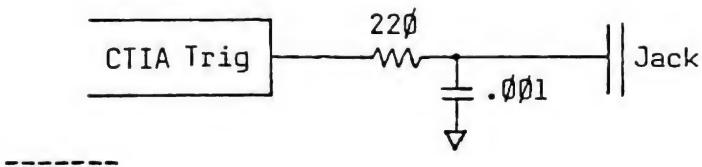
Pin 7= Vcc (+5v *)

Port B Circuit (typical):



* Note: 50ma maximum total external drain on power supply allowed

"Trigger" Port Circuit (typical):



Software Information:

6520 PIA: (this also pertains to all of the following: **)
Port A control (address \$ D302)

7 6 5 4 3 2 1 0
0 0 1 1 1 X 0 0 Write this into this register
↑—————Port A Data/Data direction addressing control
0= Address data direction at \$ D300
1= Address data at \$ D300

** Port A data direction (address \$ D300)

7 6 5 4 3 2 1 0
X X X X X X X X Write this into this register
↑↑↑↑↑↑↑↑↑↑ Data direction control for Port A
1= Out
0= In

Port A data (address \$ D300)

7 6 5 4 3 2 1 0
[] [] [] [] [] [] [] [] Read or Write this register
4 3 2 1 4 3 2 1
Jack 2 Jack 1
Pin Numbers

Port B control (address \$ D303)

0 0 1 1 1 X 0 0

6520 PIA:

Port B control (address \$ D302)

7	6	5	4	3	2	1	0
0	0	1	1	1	x	0	0

↑ Port B Data/Data direction address

Port B Data/Data direction addressing control

Ø= Address data direction

l= Address data

Port B data direction (address \$ D300)

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X
↑	↑	↑	↑	↑	↑	↑	↑

| Write this into this register

 Data direction control for Port B

l= Out

46 -

Port B data (address \$ D301)

Pin Numbers

Four "trigger" ports: (\$ D010, \$ D011, \$ D012, \$ D013)

\$ Dd1d ≡ Port 1 pin 6

\$ 0013 = Port 4 pin 6

Other miscellaneous Software information:

- 1). The O.S. sets up all PIA ports as inputs during initialization
- 2). The O.S. usually reads the above once per TV frame (during vertical blank) into RAM as follows:

Database name	Address	Data	Pins S
STICK0	0278	7 6 5 4 3 2 1 0 0 0 0 0 X X X X	Jack 1, pins 4, 3, 2, 1
STICK1	0729		Jack 2, Pins 4, 3, 2, 1
STICK2	027A		Jack 3, Pins 4, 3, 2, 1
STICK3	027B		Jack 4, Pins 4, 3, 2, 1
STRIGO	0284		Jack 1, Pin 6
STRIG1	0285	7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 X	Jack 2, Pin 6
STRIG2	0286		Jack 3, Pin 6
STRIG3	0287		Jack 4, Pin 6
PADDL1	0270	7 6 5 4 3 2 1 0 X X X X X X X X	Jack 1, Pin 5
PADDL3	0272		Jack 2, Pin 5
PADDL5	0274		Jack 3, Pin 5
PADDL7	0276		Jack 4, Pin 5
PADDL0	0271		Jack 1, Pin 9
PADDL2	0273		Jack 2, Pin 9
PADDL4	0275		Jack 3, Pin 9
PADDL 6	0277		Jack 4, Pin 9

* Pins 5 and 9 are read through the paddle controller circuitry
 a nominal value of 7 indicates that the pin is high (or floating)
 and a nominal value of 228 indicates that the pin is pulled low.

Appendix A -- CIO COMMAND BYTE values

The following hex values are known to be legitimate CIO commands.

Most handlers:

03 -- OPEN
05 -- GET RECORD
07 -- GET CHARACTERS
09 -- PUT RECORD
0B -- PUT CHARACTERS
0C -- CLOSE
0D -- GET STATUS

Display handler only:

11 -- FILL
12 -- DRAW

Disk File Manager only:

20 -- RENAME
21 -- DELETE
22 -- FORMAT
23 -- LOCK
24 -- UNLOCK
25 -- POINT
26 -- NOTE

Appendix B -- CIO STATUS BYTE values.

Shown below are the known CIO STATUS BYTE values.

01 (001) -- OPERATION COMPLETE (NO ERRORS)

80 (128) -- [BREAK] KEY ABORT
81 (129) -- IOCB ALREADY IN USE (OPEN)
82 (130) -- NON-EXISTENT DEVICE
83 (131) -- OPENED FOR WRITE ONLY
84 (132) -- INVALID COMMAND
85 (133) -- DEVICE OR FILE NOT OPEN
86 (134) -- INVALID IOCB NUMBER (Y reg only)
87 (135) -- OPENED FOR READ ONLY
88 (136) -- END OF FILE
89 (137) -- TRUNCATED RECORD
8A (138) -- DEVICE TIMEOUT (DOESN'T RESPOND)
8B (139) -- DEVICE NAK
8C (140) -- SERIAL BUS INPUT FRAMING ERROR
8D (141) -- CURSOR OUT OF RANGE
8E (142) -- SERIAL BUS DATA FRAME OVERRUN ERROR
8F (143) -- SERIAL BUS DATA FRAME CHECKSUM ERROR
90 (144) -- DEVICE DONE ERROR
91 (145) -- BAD SCREEN MODE
92 (146) -- FUNCTION NOT SUPPORTED BY HANDLER
93 (147) -- INSUFFICIENT MEMORY FOR SCREEN MODE

A0 (160) -- DISK DRIVE # ERROR
A1 (161) -- TOO MANY OPEN DISK FILES
A2 (162) -- DISK FULL
A3 (163) -- FATAL DISK I/O ERROR
A4 (164) -- INTERNAL FILE # MISMATCH
A5 (165) -- FILE NAME ERROR
A6 (166) -- POINT DATA LENGTH ERROR
A7 (167) -- FILE LOCKED
A8 (168) -- COMMAND INVALID FOR DISK
A9 (169) -- DIRECTORY FULL (64 FILES)
AA (170) -- FILE NOT FOUND
AB (171) -- POINT INVALID

Appendix C -- SIO STATUS BYTE values.

Shown below are the known SIO STATUS BYTE values.

01 (001) -- OPERATION COMPLETE (NO ERRORS)

8A (138) -- DEVICE TIMEOUT (DOESN'T RESPOND)

8B (139) -- DEVICE NAK

8C (140) -- SERIAL BUS INPUT FRAMING ERROR

8E (142) -- SERIAL BUS DATA FRAME OVERRUN ERROR

8F (143) -- SERIAL BUS DATA FRAME CHECKSUM ERROR

90 (144) -- DEVICE DONE ERROR

Appendix D -- ATASCII codes

	0X	2X	4X	6X	8X	Ax	Cx	Ex
ΦΦ	~			◊				
Φ1	!			a				
Φ2	“			b				
Φ3	”			c				
Φ4	#\$%			d				
Φ5	&			e				
Φ6	-			f				
Φ7	(g				
Φ8	*			h				
Φ9	+			i				
ΦA	,			j				
ΦB	-			k				
ΦC	.			l				
ΦD	/			m				
ΦE				n				
ΦF				o				
1Φ	Φ			p				
11	1			q				
12	2			r				
13	3			s				
14	4			t				
15	5			u				
16	6			v				
17	7			w				
18	8			x				
19	9			y				
1A	:			z				
1B	;			À				
1C	<				EOL			
1D	>				DEL LINE			
1E				CLEAR BACKSP.	JNS LINE			
1F				TAB	CLR TAB			
					SET TAB			
								BELL
								DEL CHAR
								INS CHAR

Appendix E -- Display codes (ATASCII)

	ϕX	$2X$	$4X$	$6X$	$8X$	Ax	Cx	Ex
$\phi\phi$	◆							
$\phi 1$	■							
$\phi 2$	□							
$\phi 3$	■							
$\phi 4$	■							
$\phi 5$	■							
$\phi 6$	■							
$\phi 7$	■							
$\phi 8$	■							
$\phi 9$	■							
ϕA	■							
ϕB	■							
ϕC	■							
ϕD	■							
ϕE	■							
ϕF	■							
1ϕ	♣							
11								
12								
13								
14								
15								
16								
17								
18								
19								
1A								
1B								
1C								
1D								
1E								
1F								

CODES 80-FF SHOW AS
THE INVERSE VIDEO
OF CODES 40-7F.

Appendix F --- Keyboard codes (ATASCII)

00	,	20	<space>	40	@	60	i	80-9A	/!\	00-1A00-1A
01	A	21	!	41	A	61	a	9B	<return>	& ^3
02	B	22	"	42	B	62	b	9C	s	
03	C	23	#	43	C	63	c	9D	s<insert>	
04	D	24	\$	44	D	64	d	9E	^<tab>	
05	E	25	%	45	E	65	e	9F	s<tab>	
06	F	26	&	46	F	66	f	A0-FC	/!\	20-7C
07	G	27	'	47	G	67	g	FD	^2	
08	H	28	(48	H	68	h	FE	^	
09	I	29)	49	I	69	i	FF	^<insert>	
0A	J	2A	*	4A	J	6A	j			
0B	K	2B	+	4B	K	6B	k			
0C	L	2C	,	4C	L	6C	l			
0D	M	2D	--	4D	M	6D	m			
0E	N	2E		4E	N	6E	n			
0F	O	2F	/	4F	O	6F	o			
10	P	30	0	50	P	70	p			
11	Q	31	1	51	Q	71	q			
12	R	32	2	52	R	72	r			
13	S	33	3	53	S	73	s			
14	T	34	4	54	T	74	t			
15	U	35	5	55	U	75	u			
16	V	36	6	56	V	76	v			
17	W	37	7	57	W	77	w			
18	X	38	8	58	X	78	x			
19	Y	39	9	59	Y	79	y			
1A	Z	3A	:	5A	Z	7A	z			
1B	<esc>	3B	;	5B	[7B	;			
1C	<up>	3C	<	5C	\	7C	!			
1D	<down>	3D	=	5D]	7D	<clear>			
1E	<left>	3E	>	5E	^	7E	<back>			
1F	<right>	3F	?	5F	_	7F	<tab>			

<clear> ::= s< or <
 <return> ::= <return> or s<return> or <return>
 <esc> ::= <esc> or s<esc> or <esc>
 <space> ::= <space> or s<space> or <space>

Where: s as a prefix indicates SHIFT.

^ as a prefix indicates CTRL.

/!\ as a prefix indicates ATARI key invert active.

Appendix G -- Printer codes (ATASCII)

Character set for "normal" mode printing:

20 <space>	40 @	60 `
21 !	41 A	61 a
22 "	42 B	62 b
23 #	43 C	63 c
24 \$	44 D	64 d
25 %	45 E	65 e
26 &	46 F	66 f
27 ^	47 G	67 g
28 (48 H	68 h
29)	49 I	69 i
2A *	4A J	6A j
2B +	4B K	6B k
2C ,	4C L	6C l
2D -	4D M	6D m
2E .	4E N	6E n
2F /	4F O	6F o
30 0	50 P	70 p
31 1	51 Q	71 q
32 2	52 R	72 r
33 3	53 S	73 s
34 4	54 T	74 t
35 5	55 U	75 u
36 6	56 V	76 v
37 7	57 W	77 w
38 8	58 X	78 x
39 9	59 Y	79 y
3A :	5A Z	7A z
3B ;	5B [7B {
3C <	5C \	7C
3D =	5D]	7D }
3E >	5E ^	7E ~
3F ?	5F _	7F <space>

Note: the following codes print differently than defined by the ATASCII definition.

- 00 through 1F print blank.
- 60 prints ` instead of "diamond".
- 7B prints { instead of "spade".
- 7D prints } instead of "clear".
- 7E prints ~ instead of "backspace".
- 7F prints blank instead of "tab".

Character set for "sideways" mode printing:

40 @	60 @
41 A	61 A
42 B	62 B

43	C	63	C
44	D	64	D
45	E	65	E
46	F	66	F
47	G	67	G
48	H	68	H
49	I	69	I
4A	J	6A	J
4B	K	6B	K
4C	L	6C	L
4D	M	6D	M
4E	N	6E	N
4F	O	6F	O
30	0	50	P
31	1	51	Q
32	2	52	R
33	3	53	S
34	4	54	T
35	5	55	U
36	6	56	V
37	7	57	W
38	8	58	X
39	9	59	Y
3A	:	5A	Z
3B	;	5B	£
3C	<	5C	\
3D	=	5D]
3E	>	5E <up>	7E <up>
3F	?	5F <left>	7F <left>

Note: the following codes print differently than defined by the ATASCII definition.

00 through 2F print blank.
5E prints "up arrow" instead of
5F prints "left arrow" instead of _
60 through 7F repeats 40 through 5F instead of proper set.

Appendix H -- Screen mode characteristics

Mode #	Horiz. posit.	Vert. w/o sp	Vert. w sp	Colors	Data value	Color reg.	Memory reqd.	
0	40	24	--	2	backgd. 00-FF " CO-FF	BAK PF 2 PF 1*	993	COLBK COLPFZ COLPFI
1	20	24	20	5	backgd. 00-3F 40-7F 80-BF CO-FF	BAK PF 0 PF 1 PF 2 PF 3	513	etc.
2	20	12	10	5	backgd. 00-3F 40-7F 80-BF CO-FF	BAK PF 0 PF 1 PF 2 PF 3	261	
3	40	24	20	4	0 1 2 3	BAK PF 0 PF 1 PF 2	273	
4	80	48	40	2	0 1	BAK PF 0	537	
5	80	48	40	4	0 1 2 3	BAK PF 0 PF 1 PF 2	1017	
6	160	96	80	2	0 1	BAK PF 0	2025	
7	160	96	80	4	0 1 2 3	BAK PF 0 PF 1 PF 2	3945	
8	320	192	160	2	0 1	PF 2 PF 1*	7900	
9	80	192	--	1	Note 2		7900	
10	80	192	--	9	0 1 2 3 4 5	PM 0 PM 1 PM 2 PM 3 PF 0 PF 1	7900	

6	PF 2
7	PF 3
8	BAK
9	BAK
A	BAK
B	BAK
C	PF 0
D	PF 1
E	PF 2
F	PF 3

11 80 192 --- 16 Note 3 7900

Notes:

- * Uses color of PF 2, lum of PF 1.
- 2 Uses color of BAK, lum of data value (\$0-F).
- 3 Uses color of data value (\$0-F), lum of BAK.

PF x ::= Playfield color register x.

PM x ::= Player/Missile color register x.

BAK ::= Background color register (also known as PF 4).

The default values for the color registers are shown below:

BAK	= \$00
PF0	= \$28
PF1	= \$CA
PF2	= \$94
PF3	= \$46

The form of a color register byte is shown below:

7	6	5	4	3	2	1	0
+	-	+	-	+	-	+	-
	color		lum		0		
+	-	+	-	+	-	+	-

Where: color (hex values) lum

0 = gray	0 = minimum luminance
1 = light orange	1 =
2 = orange	2 =
3 = red orange	3 = (increasing
4 = pink	4 = luminance)
5 = purple	5 =
6 = purple-blue	6 =
7 = blue	7 = maximum luminance
8 = blue	
9 = light blue	
A = turquoise	
B = green-blue	
C = green	

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

D = yellow-green

E = orange-green

F = light orange

Appendix I -- Serial Bus I.D. and command summary

Serial bus device I.D.s

Floppy disks	D1-D4	\$31-34
Printer	P1	\$40
RS-232-C	P2	\$4F
	R1-R4	\$50-53

Serial bus control codes

ACK	- \$41 ('A')
NAK	- \$4E ('N')
COMPLETE	- \$43 ('C')
ERR	- \$45 ('E')

Serial bus command codes

READ	- \$52 ('R')	Disk
WRITE	- \$57 ('W')	Printer/Disk
STATUS	- \$53 ('S')	Printer/Disk
PUT(no check)	- \$50 ('P')	Disk
FORMAT	- \$21 ('!')	Disk
DOWNLOAD	- \$20 (' ')	
READADDR	- \$54 (' ')	
READ SPIN	- \$51 (' ')	Disk
MOTOR ON	- \$55 ('V')	Disk
VERIFY SECTOR	- \$56 ('V')	Disk

Appendix J -- ROM vectors

The fixed address OS ROM JMP vectors are shown below; at each address is a JMP instruction to the indicated routine.

Name	Addr	Reference	Function
DISKIV	E450	*	Disk handler initialization
DSKINV	E453	5. 4. 2	Disk handler entry.
CIOV	E456	5. 2	CIO utility entry.
SIOV	E459	9. 3	SIO utility entry.
SETVBV	E45C	6. 7. 2	Set system timers routine.
SYSVBV	E45F	6. 3	Stage 1 VBLANK entry.
XITVBV	E462	6. 3	Exit VBLANK entry.
SIOINV	E465	*	SIO utility initialization.
SENDEV	E468	*	Send enable routine.
INTINV	E46B	*	Interrupt handler initialization.
CIOINV	E46E	*	CIO utility initialization.
BLKBDV	E471	3. 1. 1	Blackboard mode entry.
WARMSSV	E474	7.	Warmstart ([S/RESET]) entry.
COLDSV	E477	7.	Coldstart (power up) entry.

RBLOKY	E47A	*	Cassette read block entry.
CSOPIV	E47D	*	Cassette OPEN input entry.

* These vectors are for OS internal use only.

The fixed address Floating Point Package ROM routine entry point addresses are shown below; complete descriptions of the corresponding routines are provided in section 8.

AFP	D800	ASCII to FP convert.
FASC	D8E6	FP to ASCII convert.
IFP	D9AA	Integer to FP convert.
FPI	D9D2	FP to integer convert.
FADD	DA66	FP add.
FSUB	DA60	FP subtract.
FMUL	DADB	FP multiply
FDIV	DB28	FP divide.
LOG	DEC0	FP base e logarithm.
LOG10	DED1	FP base 10 logarithm.
EXP	DDC0	FP base e exponentiation.
EXP10	DDCC	FP base 10 exponentiation.
PLYEVL	DD40	FP polynomial evaluation.
ZFRO	DA44	Clear FRO.
ZF1	DA46	Clear FP number.
FLDOR	DD89	Load FP number.
FLDOP	DD8D	Load FP number.
FLD1R	DD98	Load FP number.
FLD1P	DD9C	Load FP number.
FSTOR	DDA7	Store FP number.
FSTOP	DDAB	Store FP number.
FMOVE	DDB6	Move FP number.

The base addresses of the handler vectors for the resident handlers are shown below:

Screen Editor (E)	E400
Display handler (S)	E410
Keyboard handler (K)	E420
Printer handler (P)	E430
Cassette handler (C)	E440

See section 5 for the format of the entry for each handler.

The 6502 Computer interrupt vector values are shown below:

Function	Address	Value
NMI	FFFA	E7B4
RESET	FFAC	E477
IRQ	FFFE	E6FE

APPENDIX K -- OS DATABASE VARIABLE FUNCTIONAL DESCRIPTIONS

This section contains descriptions of many of the data base variables; descriptions are included for all of the user accessible variables and for some of the "internal" variables as well. Those variables which are not considered to be normally of interest to any user are flagged with an asterisk ('*') after their names; the other variables may be of interest to one or more of the following classes of users:

- o End user.
- o Game developer.
- o Application programmer.
- o System utility writer.
- o Language processor developer.
- o Device handler writer.

Each variable is specified by its system equate file name followed by its address (in hex) and the number of bytes reserved in the data base (in decimal), in the following form:

<name> [<address>,<size>]

For example:

MEMLO [02E7,2]

Note that most word (2 byte) variables are ordered with the least significant byte at the lower address.

A. MEMORY CONFIGURATION

See section 4 for a general discussion of memory dynamics and section 7 for details of system initialization.

A1 MEMLO [02E7,2] -- User free memory low address

MEMLO contains the address of the first location in the free memory region. The value is established by the OS during power up and [RESET] initialization and is never altered by the OS thereafter.

A2 MEMTOP [02E5,2] -- User free memory high address

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

MEMTOP contains the address of the first non-useable memory location above the free memory region. The value is established by the OS during power up and [RESET] initialization; and then is re-established whenever the display is OPENed.

A3 APPMHI [000E, 2] -- User free memory screen lower limit

APPMHI is a user controlled variable which contains the address within the free memory region below which the Display handler may not go in setting up a display screen. This variable is initialized to zero by the OS at power up.

A4 RAMTOP* [006A, 1] -- Display handler top of RAM address (msb)

RAMTOP permanently retains the RAM top address that was contained in TRAMSZ (as described in N1) for the Display handler's use. The value is setup as part of handler initialization; it is not clear why this variable is required, since the same value is in RAMSIZ.

A5 RAMSIZ [02E4, 1] -- Top of RAM address (msb only)

RAMSIZ permanently retains the RAM top address that was contained in TRAMSZ (as described in N1).

B. TEXT/GRAFICS SCREEN

See section 5 for a discussion of the text and graphics screens and their handlers.

Cursor control

For the text screen and split screen text window there is a visible cursor on the screen which shows the position of the next input or output operation. The cursor is represented by inverting the video of the character upon which it resides; but the cursor may be made invisible, at the user's option. The graphics screen always has an invisible cursor.

The cursor position is sensed by examining data base variables and may be moved by altering those same variables; in addition, when using the Screen Editor, there are cursor movement control codes which may be sent as data (as explained in section 5).

B1 CRSINH [02FO, 1] -- Cursor display inhibit flag

When CRSINH is zero, all outputs to the text screen will be followed by a visible cursor (inverted character); and when CRSINH is non-zero, no visible cursor will be generated.

CRSINH is set to zero by power up, [RESET], [BREAK] or an OPEN command to the Display handler or Screen Editor.

Note that altering CRSINH does not cause the visible cursor to change states until the next output to the screen; if an immediate change to the cursor state is desired, without altering the screen data, follow the CRSINH change with the output of CURSOR UP, CURSOR DOWN or some other innocuous sequence.

B2 ROWCRS [0054, 1] & COLCRS [0055, 2] -- Current cursor position

ROWCRS and COLCRS define the cursor location (row and column, respectively) for the next data element to be read from or written to the main screen segment. When in split screen mode, the variables TXTROW and TXTCOL define the cursor for the text window at the bottom of the screen as explained in B4 below.

The row and column numbering start with the value zero, and increase monotonically to the number of rows or columns minus one; with the upper left corner of the screen being the origin (0,0).

ROWCRS is a single byte variable with a maximum allowable value of 191 (screen modes 8-11); COLCRS is a two byte variable with a maximum allowable value of 319 (screen mode 8).

B3 OLDRROW [005A, 1] & OLDCOL [005B, 2] -- Prior cursor position

OLDRROW and OLDCOL are updated from ROWCRS and COLCRS before every operation. The variables are used only for the DRAW and FILL operations.

B4 TXTROW [0290, 1] & TXTCOL [0291, 2] -- Split screen text cursor position

TXTROW and TXTCOL define the cursor location (row and column, respectively) for the next data element to be read from or written to the split screen text window.

The row and column numbering start with the value zero, and increase monotonically to 3 and 39, respectively; with the upper left corner of the split screen text window being the origin (0,0).

Screen margins

The text screen and split screen text window have user alterable left and right margins which define the normal domain of the text cursor.

B5 LMARGN [0052, 1] -- Text column left margin

LMARGN contains the column number (0-39) of the text screen left margin; the text cursor will remain on or to the right of the left margin as a result of all operations, unless the cursor column variable is directly updated by the user (see B2 and B4 above). The default value for LMARGN is 2 and is established upon power up or [RESET].

B6 RMARGN [0053, 1] -- Text column right margin

RMARGN contains the column number (0-39) of the text screen right margin; the text cursor will remain on or to the left of the right margin as a result of all operations, unless the cursor column variable is directly updated by the user (see B2 and B4 above). The default value for RMARGN is 39 and is established upon power up or [RESET].

Color control

As part of the stage 2 VBLANK process (see section 6), the values of nine data base variables are stored in corresponding hardware color control registers. The color registers are divided into two groups: the player/missile colors and the playfield colors. The playfield color registers are utilized by the different screen modes as shown in Appendix H; the player/missile color registers have no use within the standard OS.

B7 PCOLR0 - PCOLR3 [02C0, 4] -- Player/missile colors

Each color variable is stored in the corresponding hardware register as shown below:

PCOLR0 [02C0]	COLPM0 [D012]
PCOLR1 [02C1]	COLPM1 [D013]
PCOLR2 [02C2]	COLPM2 [D014]
PCOLR3 [02C3]	COLPM3 [D015]

Each color variable has the format shown below:

7	6	5	4	3	2	1	0
+	-	-	-	-	-	-	-
	color		lum	x			
+	-	-	-	-	-	-	-

See Appendix H for information regarding the color and luminance field values.

B8 COLOR0 - COLOR4 [02C5, 5] -- Playfield colors

Each color variable is stored in the corresponding hardware register as shown below:

COLOR0 [02C4]	COLPF0 [D016]
COLOR1 [02C5]	COLPF1 [D017]
COLOR2 [02C6]	COLPF2 [D018]
COLOR3 [02C7]	COLPF3 [D019]
COLOR4 [02C8]	COLBK [D01A]

Each color variable has the format shown below:

```
7 6 5 4 3 2 1 0
+---+---+---+---+---+
! color : lum !x!
+---+---+---+---+---+
```

See Appendix H for information regarding the color and luminance field values.

Text scrolling

The text screen or split screen text window "scrolls" upward whenever one of the two conditions shown below occurs:

A text line at the bottom row of the screen extends past the right margin.

A text line at the bottom row of the screen is terminated by an EOL.

Scrolling has the effect of removing the entire logical line that starts at the top of the screen and then moving all subsequent lines upward to fill in the void. The cursor will also move upward if the logical line deleted exceeds one physical line.

B9 SCRFLG* [02BB,1] --- Scroll flag

SCRFLG is a working variable that counts the number of physical lines minus one that were deleted from the top of the screen; since a logical line ranges in size from 1 to 3, SCRFLG ranges from 0 to 2.

Attract mode

Attract mode is a mechanism which protects the TV screen from having patterns "burned into" the phosphors due to a fixed display being left on the screen for extended periods of time. When the computer is left unattended for more than 9 minutes, the color intensities are limited to 50% of maximum and the hues are continually varied every 83 seconds. Pressing any keyboard data key will be sufficient to remove the attract mode for 9 more minutes.

As part of the stage 2 VBLANK process, the color registers from the data base are sent to the corresponding hardware color registers; before they are sent, they undergo the following transformation:

```
hardware register = database variable XOR COLRSH AND DRKMSK
```

Normally COLRSH = \$00 and DRKMSK = \$FE, thus making the above calculation a null operation; however, once attract mode becomes active, COLRSH = the content of RTCLOK+1 and DRKMSK = \$F6, which has the effect of modifying all of the colors and keeping their luminance always below the 50 percent level.

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

Since RTCLOK+1 is incremented every 256/60ths of a second and since the least significant bit of COLRSH is of no consequence, a color/lum change will be effected every 83 seconds (512/60).

B10 ATRACT [004D,1] -- Attract mode timer and flag

ATTRACT is the timer (and flag) which controls the initiation and termination of attract mode. Whenever a keyboard key is pressed, the keyboard IRQ service routine sets ATTRACT to zero, thus terminating attract mode; the [BREAK] key logic behaves accordingly. As part of the stage 1 VBLANK process, ATTRACT is incremented every 4 seconds; if the value exceeds 127 (after 9 minutes without keyboard activity), the value of ATTRACT will then be set to \$FE and will retain that value until attract mode is terminated.

Since the attract mode is prevented and terminated by the OS based only upon keyboard activity, some users may want to reset ATTRACT based upon ATARI controller event detection, user controlled Serial I/O bus activity or any other signs of life.

B11 COLRSH* [004F,1] -- Color shift mask

COLRSH has the value \$00 when attract mode is inactive, thus effecting no change to the screen colors; when attract mode is active, COLRSH contains the current value of the timer variable middle digit (RTCLOK+1).

B12 DRKMSK* [004E,1] -- Dark (luminance) mask

DRKMSK has the value \$FE when attract mode is inactive which does not alter the luminance; and has the value \$F6 when attract mode is active which forces the most significant bit of the luminance field to zero, thus guaranteeing that the luminance will never exceed 50 percent.

Tabbing

See section 5 for a discussion of the use of tabs in conjunction with the Screen Editor.

B13 TABMAP [02A3,15] -- Tab stop setting map

The tab settings are retained in a fifteen byte (120 bit) map, where a bit value of one indicates a tab setting; the diagram below shows the mapping of the individual bits to tab positions.

7	6	5	4	3	2	1	0											
+	-	+	-	+	-	+	-											
	0		1		2		3		4		5		6		7			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	
	8		9		10		11		12		13		14		15			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	
	=		=		=		=		=		=		=		=			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	
	11		12		13		14		15		16		17		18		19	
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	

Whenever the Display handler or Screen Editor is opened, this map is initialized to contain the value of \$01 in every byte, thus providing the default tab stops at 7, 15, 23, etc.

Logical text lines

The text screen is invisibly divided into logical lines of text, each comprising from one to three physical lines of text. The screen is initialized to 24 logical lines of one physical line each; but data entry and/or data insertion may increase the size of a logical line to two or three physical lines.

B14 LOGMAP* [02B2,4] -- Logical line starting row map

The beginning physical line number for each logical line on the screen is retained in a four-byte (32 bit) map, where a bit value of one indicates the start of a logical line; the diagram below shows the mapping of the individual bits to physical line (row) numbers.

7	6	5	4	3	2	1	0										
+	-	+	-	+	-	+	-										
	0		1		2		3		4		5		6		7		
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-
	8		9		10		11		12		13		14		15		
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-
	16		17		18		19		20		21		22		23		
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-

The map bits are all set to one whenever the text screen is OPENed or cleared. From that point, the map is updated as logical lines are entered, edited and deleted from the screen.

B15 LOGCOL* [0063,1] -- Cursor/logical line column number

LOGCOL contains the logical line column number for the current cursor position; note that a logical line may comprise up to

three physical lines. This variable is for the internal use of the Display handler.

Split screen

The Display handler and Screen Editor together support the operation of a split screen mode (see section 5) in which the main portion of the screen is in one of the graphics modes and is controlled by the Display handler, and there is a four physical line text window at the bottom of the screen which is controlled by the Screen Editor.

B16 BOTSCR* [02BF,i] -- Text screen lines count

BOTSCR contains the number of lines of text for the current screen: 24 for mode 0 or 4 for a split screen mode. The handler also uses this variable as an indication of the split screen status; tests are made for the specific values 4 and 24.

DRAW/FILL function

The DRAW function line drawing algorithm is shown below translated to the Pascal language from assembly language.

```
NEWROW := ROWCRS; NEWCOL := COLCRS;  
  
DELTAR := ABS (NEWROW-OLDROW);  
ROWINC := SIGN (NEWROW-OLDROW); { +1 or -1 }  
  
DELTAC := ABS (NEWCOL-OLDCOL);  
COLINC := SIGN (NEWCOL-OLDCOL); { +1 or -1 }  
  
ROWAC := 0; COLAC := 0;  
ROWCRS := OLDROW; COLCRS := OLDCOL;  
  
COUNTR := MAX (DELTAC, DELTAR);  
ENDPT := COUNTR;  
IF COUNTR = DELTAC  
  THEN ROWAC := ENDPT DIV 2  
  ELSE COLAC := ENDPT DIV 2;  
  
WHILE COUNTR > 0 DO  
  BEGIN  
    ROWAC := ROWAC + DELTAR;  
    IF ROWAC >= ENDPT  
      THEN  
        BEGIN  
          ROWAC := ROWAC - ENDPT;  
          ROWCRS := ROWCRS + ROWINC  
        END;
```

```
COLAC := COLAC + DELTAC;
IF COLAC >= ENDPT
  THEN
    BEGIN
      COLAC := COLAC - ENDPT;
      COLCRS := COLCRS + COLINC
    END;

    PLOT_POINT; { point defined by ROWCRS & COLCRS }

    IF FILFLG <> 0 THEN FILL_LINE;

    COUNTR := COUNTR - 1

  END;
```

The FILL function algorithm (FILL_LINE above) is described briefly in section 5.

B17 FILDAT [02FD,1] -- Fill data

FILL contains the fill region data value as part of the calling sequence for a FILL command as described in section 5.

B18 FILFLG* [02B7,1] -- Fill flag

FILFLG indicates to shared code within the Display handler whether the current operation is FILL (FILFLG <> 0) or DRAW (FILFLG = 0).

B19 NEWROW* [0060,1] & NEWCOL* [0061,2] -- Destination point

NEWROW and NEWCOL are initialized to the values in ROWCRS and COLCRS, which represent the destination endpoint of the DRAW/FILL command. This is done so that ROWCRS and COLCRS may be altered during the performance of the command.

B20 HOLD4* [02BC,1] -- Temporary storage

HOLD4 is used to save and restore the value in ATACHR during the FILL process; ATACHR is temporarily set to the value in FILDAT to accomplish the filling portion of the command.

B21 ROWINC* [0079,1] & COLINC* [007A,1] -- Row/column increment/decrement

ROWINC and COLINC are the row and column increment values; they are each set to +1 or -1 to control the basic direction of line drawing. ROWINC and COLINC represent the signs of NEWROW - ROWCRS and NEWCOL - COLCRS, respectively.

B22 DELTAR* [0076,1] & DELTAC* [0077,2] -- Delta row and delta column

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555

DELTAR and DELTAC contain the absolute values of NEWROW - ROWCRS and NEWCOL - COLCRS, respectively; together with ROWINC and COLINC, they define the slope of the line to be drawn.

B23 COUNTR* [007E,2] -- Draw iteration count

COUNTR initially contains the larger of DELTAR and DELTAC, which is the number of iterations required to generate the desired line. COUNTR is then decremented after every point on the line is plotted, until it reaches a value of zero.

B24 ROWAC* [0070,2] & COLAC* [0072,2] -- Accumulators

ROWAC and COLAC are working accumulators which control the row and column point plotting and increment (or decrement) function.

B25 ENDPT* [0074,2] -- Line length

ENDPT contains the larger of DELTAR and DELTAC, and is used in conjunction with ROWAC/COLAC and DELTAR/DELTAC to control the plotting of line points.

Displaying control characters

Often it is useful to have ATASCII control codes (such as CLEAR, CURSOR UP, etc.) displayed in their graphic forms instead of having them perform their control function. This display capability is provided in two forms when outputting to the Screen Editor: 1) a data content form in which a special character (ESC) precedes each control character to be displayed and 2) a mode control form.

Escape (display following control character)

Whenever an ESC character is detected by the Screen Editor, the next character following this code is displayed as data, even if it would normally be treated as a control code; the EOL code is the sole exception, it is always treated as a control code. The sequence ESC ESC will cause the second ESC character to be displayed.

B26 ESCFLG* [02A2,1] -- Escape flag

ESCFLG is used by the Screen Editor to control the escape sequence function; the flag is set (to \$80) by the detection of an ESC character (\$1B) in the data stream and is reset (to 0) following the output of the next character.

Display control characters mode

When it is desired to display ATASCII control codes other than EOL in their graphics form, but not have an ESC character associated with

each control code, a display mode may be established by setting a flag in the data base. This capability is used by language processors when displaying high level language statements, which may contain control codes as data elements.

B27 DSPFLG [02FE, 1] -- Display control characters flag

When DSPFLG is non-zero, ATASCII control codes other than EOL are treated as data and displayed on the screen when output to the Screen Editor. When DSPFLG is zero, ATASCII control codes are processed normally.

DSPFLG is set to zero by power up and [RESET].

Bit mapped graphics

A number of temporary variables are used by the Display handler when handling data elements (pixels) going to or from the screen; of interest here are those variables which are used to control the packing and unpacking of graphics data, where a memory byte typically contains more than one data element (for example, screen mode 8 contains 8 pixels per memory byte).

B28 DMASK* [02A0, 1] -- Pixel location mask

DMASK is a mask which contains zeroes for all bits which do not correspond to the specific pixel to be operated upon, and which contains ones for all bits which do correspond. DMASK may contain the values shown below in binary notation:

11111111	-- screen modes 1 & 2; one pixel per byte.
11110000	-- screen modes 9-11; two pixels per byte.
00001111	
11000000	-- screen modes 3, 5 & 7; four pixels per byte.
00110000	
00001100	
00000011	
10000000	-- screen modes 4, 6 & 8; eight pixels per byte.
01000000	
00000010	
00000001	

B29 SHFAMT* [006F, 1] -- Pixel justification

SHFAMT indicates the amount to shift the right justified pixel data on output, or the amount to shift the input data to right justify it on input. The value is always the same as for DMASK prior to the justification process.

Internal working variables

B30 HOLD1* [0051, 1] -- Temporary storage
B31 HOLD2* [029F, 1] -- Temporary storage
B32 HOLD3* [029D, 1] -- Temporary storage
B33 TMPCHR* [0050, 1] -- Temporary storage
B34 DSTAT* [004C, 1] -- Display status
B35 DINDEX* [0057, 1] -- Display mode

DINDEX contains the current screen mode obtained from the low order four bits of the most recent OPEN AUX1 byte.

B36 SAVMSC [0058, 2] -- Screen Memory Address

SAVMSC contains the lowest address of the screen data region; the data at that address is displayed at the upper left corner of the screen.

B37 OLDCHR* [005D, 1] -- Cursor character save/restore

OLDCHR retains the value of the character under the visible text cursor; this variable is used to restore the original character value when the cursor is moved.

B38 OLDAADR* [005E, 2] -- Cursor memory address

OLDAADR retains the memory address of the current visible text cursor location; this variable is used in conjunction with OLDCHR (B37) to restore the original character value when the cursor is moved.

B39 ADRESS* [0064, 2] -- Temporary storage

B40 MLTTMP/OPNTMP/TOADR* [0066, 2] -- Temporary storage

B41 SAVADR/FRMADR* [0068, 2] -- Temporary storage

B42 BUFCNT* [006B, 1] -- Screen Editor current logical line size

B43 BUFSTR* [006C, 2] -- Temporary storage

B44 SWPFLG* [007B, 1] --- Split screen cursor control

In split screen mode, the graphics cursor data and the text window cursor data are frequently swapped as shown below in order to get the variables associated with the region being accessed into the ROWCRS-OLDAADR variables.

ROWCRS B2 ----- TXTROW B4
COLCRS B2 ----- TXTCOL B4

DINDEX B35 ----- TINDEX B49
SAVMSC B36 ----- TXTMSC B52
OLDROW B3 ----- TXTOLD B53
OLDCOL B3 ----- " "
OLDCHR B37 ----- " "
OLDADR B38 ----- " "

SWPFLG is used to keep track of which data set is currently in the ROWCRS-OLDADR region; SWPFLG is equal to \$FF when split screen text window cursor data is in the main region, otherwise SWPFLG is equal to 0.

B45 INSDAT* [007D, 1] -- Temporary storage
B46 TMPROW* [02B8, 1] & TMPCOL* [02B9, 2] -- Temporary storage
B47 TMPLBT* [02A1, 1] -- Temporary storage
B48 SUBTMP* [029E, 1] -- Temporary storage
B49 TINDEX* [0293, 1] -- Split screen text window screen mode

TINDEX is the split screen text window equivalent of DINDEX and is always equal to zero when SWPFLG is equal to zero (see B44).

B50 BITMSK* [006E, 1] -- Temporary storage
B51 LINBUF* [0247, 40] -- Physical line buffer

LINBUF is used to temporarily buffer one physical line of text when the Screen Editor is moving screen data.

B52 TXTMSC [0294, 2] -- Splitscreen memory address

TXTMSC is the split screen text window version of SAVMSC (36).

See B44 for more information.

B53 TXTOLD* [0296, 6] -- Split screen cursor data

See B44 for more information.

Internal character code conversion

Two variables are used to retain the current character being processed (for both reading and writing); ATACHR contains the value passed to or from CIO, and CHAR contains the internal code corresponding to the value in ATACHR. Because the hardware does not interpret ATASCII characters directly, the transformations shown below are applied to all text data read and written:

ATASCII CODE	INTERNAL CODE
-----------------	------------------

00-1F	40-5F
20-3F	00-1F
40-5F	20-3F
60-7F	60-7F
80-9F	C0-DF
A0-BF	80-9F
C0-DF	A0-BF
E0-FF	E0-FF

See P26 for more information.

B54 ATACHR [02FB, 1] -- Last ATASCII character or plot point

ATACHR contains the ATASCII value for the most recent character read or written, or the value of the graphics point. This variable may also be considered to be a parameter of the FILL/DRAW commands, as the value in ATACHR will determine the line color when a DRAW or FILL is performed.

B55 CHAR* [02FA, 1] -- Internal character code

CHAR contains the internal code value for the most recent character read or written.

C. DISK HANDLER

See section 5 for a discussion of the resident Disk handler.

C1 BUFADR* [0015, 2] -- Data buffer pointer

BUFADR acts as temporary page zero pointer to the current disk buffer.

C2 DSKTIM* [0246, 1] -- Disk format operation timeout time

DSKTIM contains the timeout value for SIO calling sequence variable DTIMLO (see section 9); DSKTIM is set to 160 (which represents a 171 second timeout) at initialization time, and is updated after each disk status request operation contain the value returned in the 3rd byte of the status frame (see section 5). Note that all disk operations other than format have a fixed (7) second timeout, established by the Disk handler.

D. CASSETTE

See section 5 for a general description of the Cassette handler. The cassette uses the Serial I/O bus hardware, but does not conform with the Serial I/O bus protocol as defined in section 9. Hence, the Serial I/O utility (SIO) has cassette specific code within it. Some variables in this sub-section are utilized by SIO and some by the Cassette handler.

Cassette mode

D8 CASFLG* [030F,1] -- Cassette I/O flag

CASFLG is used internally by SIO to control the program flow through shared code. A value of zero indicates that the current operation is a standard Serial I/O bus operation, and a non-zero value indicates a cassette operation.

Cassette buffer

D9 CASBUF* [03FD,131] -- Cassette record buffer

CASBUF is the buffer used by the Cassette handler for the packing and unpacking of cassette record data, and by the initialization cassette boot logic. The format for the standard cassette record in the buffer is shown below:

7	6	5	4	3	2	1	0	
+-----+-----+-----+								
	0	1	0	1	0	1	1	CASBUF+0
+-----+-----+-----+								
	0	1	0	1	0	1	1	+1
+-----+-----+-----+								
	control	byte						+2
+-----+-----+-----+								
	128							+3
=	data		=					
	bytes							+130
+-----+-----+-----+								

See section 5 for an explanation of the standard cassette record format.

D10 BLIM* [028A,1] -- Cassette record data size

BLIM contains the count of the number of data bytes in the current cassette record being read. BLIM will have a value ranging from 1 to 128, depending upon the record control byte as explained in section 5.

D11 BPTR* [003D,1] -- Cassette record data index

BPTR contains an index into the data portion of the cassette record being read or written. The value will range from 0 to the then current value of BLIM. When BPTR equals BLIM then the buffer (CASBUF) is full if writing or empty if reading.

Internal working variables

D12 FEOF* [003F,1] -- Cassette end of file flag

FEOF is used by the cassette handler to flag the detection of an end

Appendix K -- Device Characteristics

This Appendix describes the physical characteristics of the devices that interface to the ATARI 400 and ATARI 800 Personal Computer Systems. Where applicable, data capacity, data transfer rate, storage format, SIO interface and cabling will be detailed.

Keyboard

The keyboard input rate is limited by the OS keyboard reading procedure to be 60 characters per second. The code for each key is shown in Table 5-4. A picture of the ATARI 400 Personal Computer System keyboard is shown on the following page. The keyboard hardware has no buffering and is rate limited by the debounce algorithm used.

Display

The television screen display generator has many capabilities that are not used by the Display handler (as described in Chapter 5 and shown in Appendix H); there are additional display modes, object generators, hardware display scrolling and many other features which are described in the ATARI Personal Computer System HARDWARE MANUAL.

Since all display data is stored in RAM, the display data update rate is limited primarily by the software routines that generate and format the data and access the RAM. The generation of the display from the RAM is accomplished by the ANTIC and CTIA chips using Direct Memory Access (DMA) to access the RAM data.

The internal storage formats for display data for the various modes are detailed in the ATARI Personal Computer System HARDWARE MANUAL.

ATARI 410 Program Recorder

The ATARI 410 Program Recorder has the following characteristics:

DATA CAPACITY:

xx characters per C-60 tape (unformatted).
xx characters per C-60 tape (formatted, continuous).
xx characters per C-60 tape (formatted, stop/start).

DATA TRANSFER RATES:

xx characters per second (unformatted).
xx characters per second, average (formatted, stop/start).

STORAGE FORMAT:

Tapes are recorded in 1/4 track stereo format at 1 7/8 inches per second. The tape can be recorded in both directions, where tracks 1 and 2 are side A left and right; and tracks 3 and 4 are side B right and left (industry standard). On each side, the left channel (1 or 4) is used for audio and the right channel (2 and 3) is used for digital information.

The audio channel is recorded the normal way. The digital channel is recorded using the POKEY two-tone mode producing FSK data at up to 600 baud. The MARK frequency is 5327 Hz and the SPACE frequency is 3995 Hz. The transmission of data is asynchronous byte serial as seen from the computer; POKEY reads or writes a bit serial FSK sequence for each byte, in the following order:

```
1 start bit (SPACE)
data bit-0 +-+
data bit-1 !
      +- 0 = SPACE, 1 = MARK.
data bit-6 !
data bit-7 +-+
1 stop bit (MARK)
```

The only control the computer has over tape motion is motor start/stop; and this only if the PLAY button is pressed by the user. In order for recording to take place, the user must press both the REC and PLAY buttons on the cassette. The computer has no way to sense the position of these buttons, nor even if a 410 is cabled to the computer, so the user must be careful when using this device.

SIO INTERFACE

The cassette device utilizes portions of the serial bus hardware, but does not follow any of the protocol as defined in Chapter 9.

ATARI 820 Printer

The ATARI 820 printer has the following characteristics:

DATA CAPACITY:

40 characters per line (normal printing)
29 characters per line (sideways printing)

DATA TRANSFER RATES:

Bus rate: xx characters per second.
Print time (burst): xx characters per second.
Print time (average): xx characters per second.

STORAGE FORMAT:

3 7/8 inch wide paper.
5X7 dot matrix, impact printing.

Normal format --

40 characters per line.
6 lines per inch (vertical).
12 characters per inch (horizontal).

Sideways format --

29 characters per line.
6 lines per inch (vertical).
9 characters per inch (horizontal).

SIO INTERFACE

The controller serial bus I.D. is \$40.

The controller supports the following SIO commands (see Chapter 5 for more information regarding the handler and Chapter 9 for a general discussion of bus commands):

GET STATUS

The computer sends a command frame of the format shown below:

Device I.D. = \$40.
Command byte = \$53.
Auxilliary 1 = doesn't matter.
Auxilliary 2 = doesn't matter.
Checksum = checksum of bytes above.

The printer controller responds with a data frame of the format shown in earlier in this Appendix as part of the GET STATUS discussion.

PRINT LINE

The computer sends a command frame of the format shown below:

Device I.D. = \$40.
Command byte = \$57.
Auxilliary 1 = doesn't matter.
Auxilliary 2 = \$4E for normal print or \$53 for sideways.
Checksum = checksum of bytes above.

The computer sends a data frame of the format shown below:

Leftmost character of line (column 1).
Next character of line (column 2).

Rightmost character of line (column 40 or 29).
Checksum byte.

Note that the data frame size is variable, either 41 or 30 bytes in length, depending upon the print mode specified in the command frame.

ATARI 810 Disk Drive

The ATARI 810 disk has the following characteristics:

DATA CAPACITY:

720 sectors of 128 bytes each (Disk handler format).
709 sectors of 125 data bytes each (Disk File Manager format).

DATA TRANSFER RATES:

Bus rate: xx characters per second.
Seek time: xx msec. per track + xx msec.
Rotational latency: xx msec maximum (xx rpm).

STORAGE FORMAT:

5 1/4 inch diskette, soft sectored by the controller.
40 tracks per diskette.
18 sectors per track.
128 bytes per sector.
Controlled by National INS1771-1 formatter/controller chip.
Sector interlace factor = xx.

SIO INTERFACE

The controller serial bus I.D.s range from \$31 (for 'D1') to \$34 (for 'D4').

The controller supports the following SIO commands (see earlier in this Appendix for information about the disk handler and Chapter 9 for a general discussion of bus commands):

GET STATUS

The computer sends a command frame of the format shown below:

Device I.D. = \$31-34.
Command byte = \$53.
Auxilliary 1 = doesn't matter.
Auxilliary 2 = doesn't matter.
Checksum = checksum of bytes above.

The disk controller responds with a data frame of the format shown earlier in this Appendix as part of the STATUS REQUEST discussion.

PUT SECTOR (WITH VERIFY)

The computer sends a command frame of the format shown below:

Device I. D. = \$31-34
Command byte = \$57.
Auxilliary 1 = low byte of sector number.
Auxilliary 2 = high byte of sector number (1-720).
Checksum = checksum of bytes above.

The computer sends a data frame of the format shown below:

128 data bytes.
Checksum byte.

The disk controller writes the frame data to the specified sector, then reads the sector and compares the content with the frame data. The COMPLETE byte value indicates the status of the operation.

PUT SECTOR (NO VERIFY)

The computer sends a command frame of the format shown below:

Device I. D. = \$31-34
Command byte = \$50.
Auxilliary 1 = low byte of sector number.
Auxilliary 2 = high byte of sector number (1-720).
Checksum = checksum of bytes above.

The computer sends a data frame of the format shown below:

128 data bytes.
Checksum byte.

The disk controller writes the frame data to the specified sector, then sends a COMPLETE byte value which indicates the status of the operation.

GET SECTOR

The computer sends a command frame of the format shown below:

Device I. D. = \$31-34
Command byte = \$52.
Auxilliary 1 = low byte of sector number.
Auxilliary 2 = high byte of sector number (1-720).
Checksum = checksum of bytes above.

The disk controller sends a data frame of the format shown below:

128 data bytes.
Checksum byte.

FORMAT DISK

The computer sends a command frame of the format shown below:

Operating System CO16555 -- Appendix K

Device I.D. = \$31-34
Command byte = \$21.
Auxilliary 1 = doesn't matter.
Auxilliary 2 = doesn't matter.
Checksum = checksum of bytes above.

The disk controller completely formats the disk (generates 40 tracks of 18 soft sectors per track with the data portion of each sector equal to all zeroes) and then reads each sector to verify its integrity. A data frame of 128 bytes plus checksum is returned in which the sector numbers of all bad sectors (up to a maximum of 63 sectors) are contained, followed by two consecutive bytes of \$FF. If there are no bad sectors on the disk the first two bytes of the data

Appendix L -- OS DATABASE VARIABLE FUNCTIONAL DESCRIPTIONS

Central data base description

Central data base Chapter introduction

This Appendix provides detailed information for those variables in the OS Data Base which can be altered by the user. Remaining Variables are provided narrative descriptions. Information on the variables is presented in a multiple access scheme: Lookup tables are referenced to a common set of narratives, which is itself ordered by function.

Variable descriptions are referenced by a label called a variable identifier (V.I.D.) number. The label comprises a single letter followed by a number. A different letter is assigned for each major functional area being described, and the numbers are assigned sequentially within each functional area. Those Variables which are not considered to be of interest to any user are flagged with an asterisk ('*') after their names.
The database lookup tables provided are:

1. Functional grouping -- index to the function narrative and descriptions of variables, giving VID and variable name.
2. Alphabetic list of names -- giving VID of description.
3. Address ordered list -- giving VID of description.

Item 1, the functional grouping index, starts on the next page; the other two lookup tables are at the end of Appendix L.

FUNCTIONAL INDEX TO DATABASE VARIABLE DESCRIPTIONS

A. Memory configuration

A1 MEMLO
A2 MEMTOP
A3 APPMHI
A4 RAMTOP*
A5 RAMSIZ

B. Text/graphics screen

Cursor control

B1 CRSINH
B2 ROWCRS, COLCRS
B3 OLDROW, OLDCOL
B4 TXTROW, TXTCOL

Screen margins

B5 LMARGN
B6 RMARGN

Color control

B7 PCOLR0 - PCOLR3
B8 COLOR0 - COLOR4

Text scrolling

B9 SCRFLG*

Attract mode

B10 ATRACT
B11 COLRSH*
B12 DRKMSK*

Tabbing

B13 TABMAP

Logical text lines

B14 LOGMAP*
B15 LOGCOL*

Split screen

B16 BOTSCR*

FILL/DRAW function

B17 FILDAT
B18 FILFLG*
B19 NEW 5ROW*, NEWCOL*
B20 HOLD4*
B21 ROWINC*, COLINC*
B22 DELTAR*, DELTAC*
B23 COUNTR*
B24 ROWAC*, COLAC*
B25 ENDPT*

Displaying control characters

Escape (display following control char)
B26 ESCFLG*

Display control characters mode
B27 DSPFLG

Bit mapped graphics

B28 DMASK*
B29 SHFAMT*

Internal working variables

B30 HOLD1*
B31 HOLD2*
B32 HOLD3*
B33 TMPCHR*
B34 DSTAT*
B35 DINDEX*
B36 SAVMSC
B37 OLDCHR*
B38 OLDADR*
B39 ADRESS*
B40 MLTTMP/OPNTMP/TOADR*
B41 SAVADR/FRMADR*
B42 BUFCNT*
B43 BUFSTR*
B44 SWPFLG*
B45 INSDAT*
B46 TMPROW*, TMPCOL*
B47 TMPLBT*
B48 SUBTMP*
B49 TINDEX*
B50 BITMSK*
B51 LINBUF*
B52 TXTMSC
B53 TXTOLD*

Internal character code conversion
B54 ATACHR
B55 CHAR*

C. Disk handler
C1 BUFADR*
C2 DSKTIM*

D. Cassette (part in SIO part in handler)

Baud rate determination
D1 CBAUDL*, CBAUDH*
D2 TIMFLG*
D3 TIMER1*, TIMER2*
D4 ADDCOR*
D5 TEMP1*
D6 TEMP3*
D7 SAVIO*

Cassette mode
D8 CASFLG*

Cassette buffer
D9 CASBUF*
D10 BLIM*
D11 BPTR*

Internal working variables
D12 FEOF*
D13 FTYPE*
D14 WMODE*
D15 FREQ*

E. Keyboard

Key reading and debouncing
E1 CH1*
E2 KEYDEL*
E3 CH

Special functions

Start/stop
E4 SSFLAG

[BREAK]
E5 BRKKEY

SHIFT/CONTROL lock
E6 SHFLOK
E7 HOLDCH*

Auto-repeat
E8 SRTIMR*

Inverse video
E9 INVFLG

Console switches (SELECT, START & OPTION)

F. Printer

Printer buffer

F1 PRNBUF*
F2 PBUFSZ*
F3 PBPNT*

Internal working variables
F4 PTEMP*
F5 PTIMOT*

G. Central I/O routine (CIO)

User call parameters

G1 IOCB
G2 ICHID
G3 ICDNO
G4 ICCOM
G5 ICSTA
G6 ICBAL, ICBAH
G7 ICPTL, ICPTH
G8 ICBLL, ICBLH
G9 ICAX1, ICAX2
G10 ICSPR

Device status
G11 DVSTAT

Device Table
G12 HATABS

CIO/handler interface parameters

G13 ZIOCB (IOCBAS)
G14 ICHIDZ
G15 ICDNOZ
G16 ICCOMZ
G17 ICSTAZ
G18 ICBALZ, ICBALH
G19 ICPTLZ, ICPTHZ
G20 ICBLLZ, ICBLHZ
G21 ICAX1Z, ICAX2Z
G22 ICSPRZ (ICIDNO, CIOCHR)

Internal working variables

G23 ICCOMT*
G24 ICIDNO*
G25 CIOCHR*

H. Serial I/O routine (SIO)

User call parameters

H1 DCB control block
H2 DDEVIC
H3 DUNIT
H4 DCOMND
H5 DSTATS
H6 DBUFLO, DBUFHI
H7 DTIMLO
H8 DBYTLO, DBYTHI
H9 DAUX1, DAUX2

Bus sound control
H10 SOUNDR

Serial bus control

Retry logic
H11 CRETRY*
H12 DRETRY*

Checksum
H13 CHKSUM*
H14 CHKSNT*
H15 NOCKSM*

Data buffering

General buffer control

H16 BUFRLO*, BUFRHI*
H17 BFENLO*, BFENHI*

Command frame output buffer

H18 CDEVIC*
H19 CCOMND*
H20 CAUX1*, CAUX2*

Receive/transmit data buffering

H21 BUFRFL*
H22 RECVDN*
H23 TEMP*
H24 XMTDON*

SIO timeout

H25 TIMFLG*
H26 CDTMV1*
H27 CDTMA1*

Internal working variables

H28 STACKP*
H29 TSTAT*
H30 ERRFLG*
H31 STATUS*
H32 SSKCTL*

J. ATARI controllers

Joysticks

J1 STICK0 - STICK3
J2 STRIGO - STRIG3

Paddles

J3 PADDL0 - PADDL7
J4 PTRIG0 - PTRIG7

Light pen

J5 LPENH
J6 LPENV
J7 STICK0

Driving controllers

J8 STICK0 - STICK3
J9 STRIGO - STRIG3

K. Disk file manager

K1 FMSZPG*
K2 ZBUFP*
K3 ZDRVIA*
K4 ZSBA*
K5 ERRNO*

L. Disk utilities (DOS)

L1 DSKUTL*

M. Floating point package

M1 FRO
M2 FRE*
M3 FR1
M4 FR2*
M5 FRX*
M6 EEXP*
M7 NSIGN*
M8 ESIGN*
M9 FCHRFLG*
M10 DIGRT*
M11 CIX
M12 INBUFF
M13 ZTEMP1*
M14 ZTEMP4*
M15 ZTEMP3*
M16 FLPTR
M17 FPTR2*
M18 LBPR1*
M19 LBPR2*
M20 LBUFF
M21 PLYARG*
M22 FPSCR/FSCR*
M23 FPSCR1/FSCR1*
M24 DEGFLG/RADFLG*

N. Power up & [S RESET]

RAM sizing
N1 RAMLO*, TRAMSZ*
N2 TSTDAT*

Disk/cassette boot

N3 DOSINI
N4 CKEY*
N5 CASBT*
N6 CASINI
N7 BOOT?*
N8 DFLAGS*
N9 DBSECT*
N10 BOOTAD*

Environmental control

N11 COLDST*
N12 DOSVEC

[S RESET]

N13 WARMST

P. Interrupts

P1 CRITIC
P2 POKMSK

System timers

Real-time clock
P3 RTCLOK

System timer 1
P4 CDTMV1
P5 CDTMA1

System timer 2
P6 CDTMV2
P7 CDTMA2

System timers 3-5
P8 CDTMV3, CDTMV4, CDTMV5
P9 CDTMF3, CDTMF4, CDTMF5

RAM interrupt vectors

NMI interrupt vectors
P10 VDSLST
P11 VVBLKI
P12 VVBLKD

IRQ interrupt vectors
P13 VIMIRQ
P14 VPRCED
P15 VINTER
P16 VBREAK
P17 VKEYBD
P18 VSERIN
P19 VSEROR
P20 VSEROC
P21 VTIMR1, VTIMR2, VTIMR4

Hardware register updates
P22 SDMCTL*
P23 SDLSTL*, SDLSTH*
P24 GPRIOR*
P25 CHACT*
P26 CHBAS
P27 PCOLRx, COLORx

Internal working variable
P28 INTEMP*

R. User areas

R1 (unlabeled)
R2 USAREA

S. Unused (spare) bytes

S1 HOLDS
S2 CSTAT
S3 DUNUSE
S4 TEMP2
S5 TMPX1
S6 DSKFMS
S7-S15 (unlabeled)

This Appendix contains descriptions of many of the data base variables; descriptions are included for all of the user accessible variables and for some of the "internal" variables as well. Those variables which are not considered to be normally of interest to any user are flagged with an asterisk (*) after their names; the other variables may be of interest to one or more of the following classes of users:

- o End user.
- o Game developer.

- o Application programmer.
- o System utility writer.
- o Language processor developer.
- o Device handler writer.

Each variable is specified by its system equate file name followed by its address (in hex) and the number of bytes reserved in the data base (in decimal), in the following form:

<name> [<address>, <size>]

For example:

MEMLO [02E7, 2]

Note that most word (2 byte) variables are ordered with the least significant byte at the lower address.

A. MEMORY CONFIGURATION

See Chapter 4 for a general discussion of memory dynamics and Chapter 7 for details of system initialization.

A1 MEMLO [02E7, 2] -- User free memory low address

MEMLO contains the address of the first location in the free memory region. The value is established by the OS during power up and [RESET] initialization and is never altered by the OS thereafter.

A2 MEMTOP [02E5, 2] -- User free memory high address

MEMTOP contains the address of the first non-useable memory location above the free memory region. The value is established by the OS during power up and [RESET] initialization; and then is re-established whenever the display is OPENed, based upon the requirements of the selected graphics mode.

A3 APPMHI [000E, 2] -- User free memory screen lower limit

APPMHI is a user controlled variable which contains the address within the free memory region below which the Display handler may not go in setting up a display screen. This variable is initialized to zero by the OS at power up.

A4 RAMTOP* [006A, 1] -- Display handler top of RAM address (msb)

RAMTOP permanently retains the RAM top address that was contained in TRAMSZ (as described in N1) for the Display handler's use. The value

is setup as part of handler initialization; it is not clear why this variable is required, since the same value is in RAMSIZ.

A5 RAMSIZ [02E4,1] -- Top of RAM address (msb only)

RAMSIZ permanently retains the RAM top address that was contained in TRAMSIZ (as described in N1).

B. TEXT/GRAFICS SCREEN

See Chapter 5 for a discussion of the text and graphics screens and their handlers.

Cursor control

For the text screen and split screen text window there is a visible cursor on the screen which shows the position of the next input or output operation. The cursor is represented by inverting the video of the character upon which it resides; but the cursor may be made invisible, at the user's option. The graphics screen always has an invisible cursor.

The cursor position is sensed by examining data base variables and may be moved by altering those same variables; in addition, when using the Screen Editor, there are cursor movement control codes which may be sent as data (as explained in Chapter 5).

B1 CRSINH [02F0,1] -- Cursor display inhibit flag

When CRSINH is zero, all outputs to the text screen will be followed by a visible cursor (inverted character); and when CRSINH is non-zero, no visible cursor will be generated.

CRSINH is set to zero by power up, [RESET], [BREAK] or an OPEN command to the Display handler or Screen Editor.

Note that altering CRSINH does not cause the visible cursor to change states until the next output to the screen; if an immediate change to the cursor state is desired, without altering the screen data, follow the CRSINH change with the output of CURSOR UP, CURSOR DOWN or some other innocuous sequence.

B2 ROWCRS [0054,1] & COLCRS [0055,2] -- Current cursor position

ROWCRS and COLCRS define the cursor location (row and column, respectively) for the next data element to be read from or written to the main screen segment. When in split screen mode, the variables TXTROW and TXTCOL define the cursor for the text window at the bottom of the screen as explained in B4 below.

The row and column numbering start with the value zero, and increase monotonically to the number of rows or columns minus one; with the upper left corner of the screen being the origin (0,0).

ROWCRS is a single byte variable with a maximum allowable value of 191 (screen modes 8-11); COLCRS is a two byte variable with a maximum allowable value of 319 (screen mode 8).

B3 OLDRW [005A,1] & OLDCOL [005B,2] -- Prior cursor position

OLDRW and OLDCOL are updated from ROWCRS and COLCRS before every operation. The variables are used only for the DRAW and FILL operations.

B4 TXTROW [0290,1] & TXTCOL [0291,2] -- Split screen text cursor position

TXTROW and TXTCOL define the cursor location (row and column, respectively) for the next data element to be read from or written to the split screen text window.

The row and column numbering start with the value zero, and increase monotonically to 3 and 39, respectively; with the upper left corner of the split screen text window being the origin (0,0).

Screen margins

The text screen and split screen text window have user alterable left and right margins which define the normal domain of the text cursor.

B5 LMARGN [0052,1] -- Text column left margin

LMARGN contains the column number (0-39) of the text screen left margin; the text cursor will remain on or to the right of the left margin as a result of all operations, unless the cursor column variable is directly updated by the user (see B2 and B4 above). The default value for LMARGN is 2 and is established upon power up or [RESET].

B6 RMARGN [0053,1] -- Text column right margin

RMARGN contains the column number (0-39) of the text screen right margin; the text cursor will remain on or to the left of the right margin as a result of all operations, unless the cursor column variable is directly updated by the user (see B2 and B4 above). The default value for RMARGN is 39 and is established upon power up or [RESET].

Color control

As part of the stage 2 VBLANK process (see Chapter 6), the values of nine data base variables are stored in corresponding hardware color

control registers. The color registers are divided into two groups: the player/missile colors and the playfield colors. The playfield color registers are utilized by the different screen modes as shown in Appendix H; the player/missile color registers have no use within the standard OS.

B7 PCOLR0 - PCOLR3 [02C0, 4] -- Player/missile colors

Each color variable is stored in the corresponding hardware register as shown below:

PCOLR0 [02C0]	COLPM0 [D012]
PCOLR1 [02C1]	COLPM1 [D013]
PCOLR2 [02C2]	COLPM2 [D014]
PCOLR3 [02C3]	COLPM3 [D015]

Each color variable has the format shown below:

7	6	5	4	3	2	1	0
+	-	-	-	-	-	-	-
:	color	:	lum	:	x	:	
+	-	-	-	-	-	-	-

See Appendix H for information regarding the color and luminance field values.

B8 COLOR0 - COLOR4 [02C5, 5] -- Playfield colors

Each color variable is stored in the corresponding hardware register as shown below:

COLOR0 [02C4]	COLPFO [D016]
COLOR1 [02C5]	COLPF1 [D017]
COLOR2 [02C6]	COLPF2 [D018]
COLOR3 [02C7]	COLPF3 [D019]
COLOR4 [02C8]	COLBK [D01A]

Each color variable has the format shown below:

7	6	5	4	3	2	1	0
+	-	-	-	-	-	-	-
:	color	:	lum	:	x	:	
+	-	-	-	-	-	-	-

See Appendix H for information regarding the color and luminance field values.

Text scrolling

The text screen or split screen text window "scrolls" upward whenever one of the two conditions shown below occurs:

A text line at the bottom row of the screen extends past the right margin.

A text line at the bottom row of the screen is terminated by an EOL.

Scrolling has the effect of removing the entire logical line that starts at the top of the screen and then moving all subsequent lines upward to fill in the void. The cursor will also move upward if the logical line deleted exceeds one physical line.

B9 SCRFLG* [02BB, 1] -- Scroll flag

SCRFLG is a working variable that counts the number of physical lines minus one that were deleted from the top of the screen; since a logical line ranges in size from 1 to 3, SCRFLG ranges from 0 to 2.

Attract mode

Attract mode is a mechanism which protects the TV screen from having patterns "burned into" the phosphors due to a fixed display being left on the screen for extended periods of time. When the computer is left unattended for more than 9 minutes, the color intensities are limited to 50% of maximum and the hues are continually varied every 8.3 seconds. Pressing any keyboard data key will be sufficient to remove the attract mode for 9 more minutes.

As part of the stage 2 VBLANK process, the color registers from the data base are sent to the corresponding hardware color registers; before they are sent, they undergo the following transformation:

hardware register = database variable XOR COLRSH AND DRKMSK

Normally COLRSH = \$00 and DRKMSK = \$FE, thus making the above calculation a null operation; however, once attract mode becomes active, COLRSH = the content of RTCLOK+1 and DRKMSK = \$F6, which has the effect of modifying all of the colors and keeping their luminance always below the 50 percent level.

Since RTCLOK+1 is incremented every 256/60ths of a second and since the least significant bit of COLRSH is of no consequence, a color/lum change will be effected every 8.3 seconds (512/60).

B10 ATRACT [004D, 1] -- Attract mode timer and flag

ATRACT is the timer (and flag) which controls the initiation and termination of attract mode. Whenever a keyboard key is pressed, the keyboard IRQ service routine sets ATRACT to zero, thus terminating attract mode; the [BREAK] key logic behaves accordingly. As part of the stage 1 VBLANK process, ATRACT is incremented every 4 seconds; if the value exceeds 127 (after 9 minutes without keyboard activity), the value of ATRACT will then be set to \$FE and will retain that value until attract mode is terminated.

Since the attract mode is prevented and terminated by the OS based only upon keyboard activity, some users may want to reset ATTRACT based upon ATARI controller event detection, user controlled Serial I/O bus activity or any other signs of life.

B11 COLRSH* [C004F, 1] -- Color shift mask

COLRSH has the value \$00 when attract mode is inactive, thus effecting no change to the screen colors; when attract mode is active, COLRSH contains the current value of the timer variable middle digit (RTCLK+1).

B12 DRKMSK* [C004E, 1] -- Dark (luminance) mask

DRKMSK has the value \$FE when attract mode is inactive which does not alter the luminance; and has the value \$F6 when attract mode is active which forces the most significant bit of the luminance field to zero, thus guaranteeing that the luminance will never exceed 50 percent.

Tabbing

See Chapter 5 for a discussion of the use of tabs in conjunction with the Screen Editor.

B13 TABMAP [C02A3, 15] -- Tab stop setting map

The tab settings are retained in a fifteen byte (120 bit) map, where a bit value of one indicates a tab setting; the diagram below shows the mapping of the individual bits to tab positions.

7	6	5	4	3	2	1	0										
:	0	:	1	:	2	:	3	:	4	:	5	:	6	:	7	:	TABMAP+0
+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
:	8	:	9	:	10	:	11	:	12	:	13	:	14	:	15		+1
+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
:																	
=																	
:																	
+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
:	112	:	113	:	114	:	115	:	116	:	117	:	118	:	119		+14
+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Whenever the Display handler or Screen Editor is opened, this map is initialized to contain the value of \$01 in every byte, thus providing the default tab stops at 7, 15, 23, etc.

Logical text lines

The text screen is invisibly divided into logical lines of text, each comprising from one to three physical lines of text. The screen is initialized to 24 logical lines of one physical line

each; but data entry and/or data insertion may increase the size of a logical line to two or three physical lines.

B14 LOGMAP* [02B2,4] -- Logical line starting row map

The beginning physical line number for each logical line on the screen is retained in a four byte (32 bit) map, where a bit value of one indicates the start of a logical line; the diagram below shows the mapping of the individual bits to physical line (row) numbers.

7	6	5	4	3	2	1	0										
+	-	-	-	-	-	-	-	+									
	0	:	1	:	2	:	3	:	4	:	5	:	6	:	7	:	LOGMAP+0
+	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	+	
	8	:	9	:	10	:	11	:	12	:	13	:	14	:	15	:	+1
+	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	+	
	16	:	17	:	18	:	19	:	20	:	21	:	22	:	23	:	+2
+	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	+	
																	+3
+	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	+	

The map bits are all set to one whenever the text screen is OPENed or cleared. From that point, the map is updated as logical lines are entered, edited and deleted from the screen.

B15 LOGCOL* [0063,1] -- Cursor/logical line column number

LOGCOL contains the logical line column number for the current cursor position; note that a logical line may comprise up to three physical lines. This variable is for the internal use of the Display handler.

Split screen

The Display handler and Screen Editor together support the operation of a split screen mode (see Chapter 5) in which the main portion of the screen is in one of the graphics modes and is controlled by the Display handler, and there is a four physical line text window at the bottom of the screen which is controlled by the Screen Editor.

B16 BOTSCR* [02BF,1] -- Text screen lines count

BOTSCR contains the number of lines of text for the current screen: 24 for mode 0 or 4 for a split screen mode. The handler also uses this variable as an indication of the split screen status; tests are made for the specific values 4 and 24.

DRAW/FILL function

The DRAW function line drawing algorithm is shown below translated to the Pascal language from assembly language.

```
NEWROW := ROWCRS; NEWCOL := COLCRS;

DELTAR := ABS (NEWROW-OLDROW);
ROWINC := SIGN (NEWROW-OLDROW); { +1 or -1 }

DELTAC := ABS (NEWCOL-OLDCOL);
COLINC := SIGN (NEWCOL-OLDCOL); { +1 or -1 }

ROWAC := 0; COLAC := 0;
ROWCRS := OLDROW; COLCRS := OLDCOL;

COUNTR := MAX (DELTAC, DELTAR);
ENDPT := COUNTR;
IF COUNTR = DELTAC
    THEN ROWAC := ENDPT DIV 2
    ELSE COLAC := ENDPT DIV 2;

WHILE COUNTR > 0 DO
BEGIN

    ROWAC := ROWAC + DELTAR;
    IF ROWAC >= ENDPT
        THEN
            BEGIN
                ROWAC := ROWAC - ENDPT;
                ROWCRS := ROWCRS + ROWINC
            END;

    COLAC := COLAC + DELTAC;
    IF COLAC >= ENDPT
        THEN
            BEGIN
                COLAC := COLAC - ENDPT;
                COLCRS := COLCRS + COLINC
            END;

    PLOT_POINT; { point defined by ROWCRS & COLCRS }

    IF FILFLG <> 0 THEN FILL_LINE;

    COUNTR := COUNTR - 1

END;
```

The FILL function algorithm (FILL_LINE above) is described briefly in Chapter 5.

B17 FILDAT [02FD,1] -- Fill data

FILL contains the fill region data value as part of the calling sequence for a FILL command as described in Chapter 5.

B18 FILFLG* [02B7,1] -- Fill flag

FILFLG indicates to shared code within the Display handler whether the current operation is FILL (FILFLG \neq 0) or DRAW (FILFLG = 0).

B19 NEWROW* [0060,1] & NEWCOL* [0061,2] -- Destination point

NEWROW and NEWCOL are initialized to the values in ROWCRS and COLCRS, which represent the destination endpoint of the DRAW/FILL command. This is done so that ROWCRS and COLCRS may be altered during the performance of the command.

B20 HOLD4* [02BC,1] -- Temporary storage

HOLD4 is used to save and restore the value in ATACHR during the FILL process; ATACHR is temporarily set to the value in FILDAT to accomplish the filling portion of the command.

B21 ROWINC* [0079,1] & COLINC* [007A,1] -- Row/column increment/decrement

ROWINC and COLINC are the row and column increment values; they are each set to +1 or -1 to control the basic direction of line drawing. ROWINC and COLINC represent the signs of NEWROW - ROWCRS and NEWCOL - COLCRS, respectively.

B22 DELTAR* [0076,1] & DELTAC* [0077,2] -- Delta row and delta column

DELTAR and DELTAC contain the absolute values of NEWROW - ROWCRS and NEWCOL - COLCRS, respectively; together with ROWINC and COLINC, they define the slope of the line to be drawn.

B23 COUNTR* [007E,2] -- Draw iteration count

COUNTR initially contains the larger of DELTAR and DELTAC, which is the number of iterations required to generate the desired line. COUNTR is then decremented after every point on the line is plotted, until it reaches a value of zero.

B24 ROWAC* [0070,2] & COLAC* [0072,2] -- Accumulators

ROWAC and COLAC are working accumulators which control the row and column point plotting and increment (or decrement) function.

B25 ENDPT* [0074,2] -- Line length

ENDPT contains the larger of DELTAR and DELTAC, and is used in conjunction with ROWAC/COLAC and DELTAR/DELTAC to control the plotting of line points.

Displaying control characters

Often it is useful to have ATASCII control codes (such as CLEAR, CURSOR UP, etc.) displayed in their graphic forms instead of having

them perform their control function. This display capability is provided in two forms when outputting to the Screen Editor: 1) a data content form in which a special character (ESC) precedes each control character to be displayed and 2) a mode control form.

Escape (display following control character)

Whenever an ESC character is detected by the Screen Editor, the next character following this code is displayed as data, even if it would normally be treated as a control code; the EOL code is the sole exception, it is always treated as a control code. The sequence ESC ESC will cause the second ESC character to be displayed.

B26 ESCFLG* [02A2, 1] -- Escape flag

ESCFLG is used by the Screen Editor to control the escape sequence function; the flag is set (to \$80) by the detection of an ESC character (\$1B) in the data stream and is reset (to 0) following the output of the next character.

Display control characters mode

When it is desired to display ATASCII control codes other than EOL in their graphics form, but not have an ESC character associated with each control code, a display mode may be established by setting a flag in the data base. This capability is used by language processors when displaying high level language statements, which may contain control codes as data elements.

B27 DSPFLG [02FE, 1] -- Display control characters flag

When DSPFLG is non-zero, ATASCII control codes other than EOL are treated as data and displayed on the screen when output to the Screen Editor. When DSPFLG is zero, ATASCII control codes are processed normally.

DSPFLG is set to zero by power up and [RESET].

Bit mapped graphics

A number of temporary variables are used by the Display handler when handling data elements (pixels) going to or from the screen; of interest here are those variables which are used to control the packing and unpacking of graphics data, where a memory byte typically contains more than one data element (for example, screen mode 8 contains 8 pixels per memory byte).

B28 DMASK* [02A0, 1] -- Pixel location mask

DMASK is a mask which contains zeroes for all bits which do not correspond to the specific pixel to be operated upon, and which contains ones for all bits which do correspond. DMASK may contain the values shown below in binary notation:

```
11111111 -- screen modes 1 & 2; one pixel per byte.  
11110000 -- screen modes 9-11; two pixels per byte.  
00001111  
  
11000000 -- screen modes 3, 5 & 7; four pixels per byte.  
00110000  
00001100  
00000011  
  
10000000 -- screen modes 4, 6 & 8; eight pixels per byte.  
01000000  
  
00000010  
00000001
```

B29 SHFAMT* [006F,1] -- Pixel justification

SHFAMT indicates the amount to shift the right justified pixel data on output, or the amount to shift the input data to right justify it on input. The value is always the same as for DMASK prior to the justification process.

Internal working variables

B30 HOLD1* [0051,1] -- Temporary storage
B31 HOLD2* [029F,1] -- Temporary storage
B32 HOLD3* [029D,1] -- Temporary storage
B33 TMPCHR* [0050,1] -- Temporary storage
B34 DSTAT* [004C,1] -- Display status
B35 DINDEX* [0057,1] -- Display mode

DINDEX contains the current screen mode obtained from the low order four bits of the most recent OPEN AUX1 byte.

B36 SAVMSC [0058,2] -- Screen Memory Address

SAVMSC contains the lowest address of the screen data region; the data at that address is displayed at the upper left corner of the screen.

B37 OLDCHR* [005D,1] -- Cursor character save/restore

OLDCHR retains the value of the character under the visible text cursor; this variable is used to restore the original character value when the cursor is moved.

B38 OLDADR* [005E,2] -- Cursor memory address

OLDADR retains the memory address of the current visible text cursor location; this variable is used in conjunction with OLDCHR (B37) to restore the original character value when the cursor is moved.

B39 ADRESS* [0064,2] -- Temporary storage

B40 MLTTMP/OPNTMP/TOADR* [0066,2] -- Temporary storage

B41 SAVADR/FRMADR* [0068,2] -- Temporary storage

B42 BUFCNT* [006B,1] -- Screen Editor current logical line size

B43 BUFSTR* [006C,2] -- Temporary storage

B44 SWPFLG* [007B,1] -- Split screen cursor control

In split screen mode, the graphics cursor data and the text window cursor data are frequently swapped as shown below in order to get the variables associated with the region being accessed into the ROWCRS-OLDADR variables.

ROWCRS B2 -----	TXTROW B4
COLCRS B2 -----	TXTCOL B4
DINDEX B35 -----	TINDEX B49
SAVMSC B36 -----	TXTMSC B52
OLDROW B3 -----	TXTOLD B53
OLDCOL B3 -----	" "
OLDCHR B37 -----	" "
OLDADR B38 -----	" "

SWPFLG is used to keep track of which data set is currently in the ROWCRS-OLDADR region; SWPFLG is equal to \$FF when split screen text window cursor data is in the main region, otherwise SWPFLG is equal to 0.

B45 INSDAT* [007D,1] -- Temporary storage

B46 TMPROW* [02B8,1] & TMPCOL* [02B9,2] -- Temporary storage

B47 TMPLBT* [02A1,1] -- Temporary storage

B48 SUBTMP* [029E,1] -- Temporary storage

B49 TINDEX* [0293,1] -- Split screen text window screen mode

TINDEX is the split screen text window equivalent of DINDEX and is always equal to zero when SWPFLG is equal to zero (see B44).

B50 BITMSK* [006E,1] -- Temporary storage

B51 LINBUF* [0247,40] -- Physical line buffer

LINBUF is used to temporarily buffer one physical line of text when the Screen Editor is moving screen data.

B52 TXTMSC [0294,2] -- Split screen memory address

TXTMSC is the split screen text window version of SAVMSC (B36).

See B44 for more information.

B53 TXTOLD* [0296,6] -- Split screen cursor data

See B44 for more information.

Internal character code conversion

Two variables are used to retain the current character being processed (for both reading and writing); ATACHR contains the value passed to or from CIO, and CHAR contains the internal code corresponding to the value in ATACHR. Because the hardware does not interpret ATASCII characters directly, the transformations shown below are applied to all text data read and written:

ATASCII CODE	INTERNAL CODE
00-1F	40-5F
20-3F	00-1F
40-5F	20-3F
60-7F	60-7F
80-9F	C0-DF
A0-BF	80-9F
C0-DF	A0-BF
E0-FF	E0-FF

See P26 for more information.

B54 ATACHR [02FB,1] -- Last ATASCII character or plot point

ATACHR contains the ATASCII value for the most recent character read or written, or the value of the graphics point. This variable may also be considered to be a parameter of the FILL/DRAW commands, as the value in ATACHR will determine the line color when a DRAW or FILL is performed.

B55 CHAR* [02FA,1] -- Internal character code

CHAR contains the internal code value for the most recent character read or written.

C. DISK HANDLER

See Chapter 5 for a discussion of the resident Disk handler.

C1 BUFADR* [0015,2] -- Data buffer pointer

BUFADR acts as temporary page zero pointer to the current disk buffer.

C2 DSKTIM* [0246,1] -- Disk format operation timeout time

DSKTIM contains the timeout value for SIO calling sequence variable DTIMLO (see Chapter 9); DSktim is set to 160 (which represents a 171 second timeout) at initialization time, and is updated after each disk status request operation contain the value returned in the 3rd byte of the status frame (see Chapter 5). Note that all disk operations other than format have a fixed (7) second timeout, established by the Disk handler.

D. CASSETTE

See Chapter 5 for a general description of the Cassette handler. The cassette uses the Serial I/O bus hardware, but does not conform with the Serial I/O bus protocol as defined in Chapter 9. Hence, the Serial I/O utility (SIO) has cassette specific code within it. Some variables in this sub-Chapter are utilized by SIO and some by the Cassette handler.

Baud rate determination

The input baud rate is assumed to be a nominal 600 baud, but will be adjusted, if necessary, by the SIO routine to account for drive motor variations, stretched tape, etc. The beginning of every cassette record contains a pattern of alternating ones and zeroes which is used solely for speed correction; by measuring the time to read a fixed number of bits, the true receive baud rate is determined and the hardware adjusted accordingly. Input baud rates ranging from 318 to 1407 baud can theoretically be handled using this technique.

The input baud rate is adjusted by setting the POKEY counter which controls the bit sampling period.

D1 CBAUDL* [02EE,1] & CBAUDH* [02EF,1] -- Cassette baud rate

Initialized to 05CC hex, which represents a nominal 600 baud. After baud rate calculation, these variables will contain POKEY counter values for the corrected baud rate.

D2 TIMFLG* [0317,1] -- Baud rate determination time out flag

TIMFLG is used by SIO to timeout an unsuccessful baud rate determination. The flag is initially set to one, and if it attains a value of zero (after 2 seconds) before the first byte of the cassette record has been read, the operation will be aborted. See also H24.

D3 TIMER1* [030C,2] & TIMER2* [0310,2] -- Baud rate timers

These timers contain reference times for the beginning and end of the fixed bit pattern receive period. The first byte of each timer contains the then current vertical line counter value read from ANTIC, and the second byte of each timer contains the then current value of the least significant byte of the OS real time clock (RTCLOK+2).

The difference between the timers is converted to raster pair counts and is then used to perform a table lookup with interpolation to determine the new values for CBAUDL and CBAUDH.

D4 ADDCOR* [030E,1] -- Interpolation adjustment variable

ADDCOR is a temporary variable used for the interpolation calculation of the above computation.

D5 TEMP1* [0312,2] -- Temporary storage

D6 TEMP3* [0315,1] -- Temporary storage

D7 SAVIO* [0316,1] -- Serial in data detect

SAVIO is used to retain the state of SKSTAT [D20F] bit-4 (serial data in); it is used to detect (and is updated after) every bit arrival.

Cassette mode

D8 CASFLG* [030F,1] -- Cassette I/O flag

CASFLG is used internally by SIO to control the program flow through shared code. A value of zero indicates that the current operation is a standard Serial I/O bus operation, and a non-zero value indicates a cassette operation.

Cassette buffer

D9 CASBUF* [03FD,131] -- Cassette record buffer

CASBUF is the buffer used by the Cassette handler for the packing and unpacking of cassette record data, and by the initialization cassette boot logic. The format for the standard cassette record in the buffer is shown below:

7 6 5 4 3 2 1 0	
+---+---+---+---+---+	
: 0 1 0 1 0 1 0 1 :	CASBUF+0
+---+---+---+---+---+	
: 0 1 0 1 0 1 0 1 :	+1
+---+---+---+---+---+	
: control byte :	+2
+---+---+---+---+---+	
: 128 :	+3
= data =	
: bytes :	+130
+---+---+---+---+---+	

See Chapter 5 for an explanation of the standard cassette record format.

D10 BLIM* [028A,1] -- Cassette record data size

BLIM contains the count of the number of data bytes in the current cassette record being read. BLIM will have a value ranging from 1 to 128, depending upon the record control byte as explained in Chapter 5.

D11 BPTR* [003D,1] -- Cassette record data index

BPTR contains an index into the data portion of the cassette record being read or written. The value will range from 0 to the then current value of BLIM. When BPTR equals BLIM then the buffer (CASBUF) is full if writing or empty if reading.

Internal working variables

D12 FEOF* [003F,1] -- Cassette end of file flag

FEOF is used by the cassette handler to flag the detection of an end of file condition (control byte = \$FE). FEOF equal to zero indicates that an EOF has not yet been detected, and a non-zero value indicates that an EOF has been detected. The flag is reset at every OPEN.

D13 FTYPE* [003E,1] -- Inter-record gap type

FTYPE is a copy of ICAX2Z from the OPEN command and indicates the type of inter-record gap selected; a positive value indicates normal record gaps, and a negative value indicates continuous mode gaps.

D14 WMODE* [0289,1] -- Cassette read/write mode flag

WMODE is used by the cassette handler to indicate whether the current operation is a read or write operation; a value of zero indicates read, and a value of \$80 indicates write.

D15 FREQ* [0040,1] -- Beep count

FREQ is used to retain and count the number of beeps requested of the 'BEEP' routine by the Cassette handler during the OPEN command process.

E. KEYBOARD

See Chapter 5 for a general description of the Keyboard handler.

Key reading and debouncing

The console key code register is read in response to an IRQ interrupt which is generated whenever a key stroke is detected by the hardware. The key code is compared with the prior key code accepted (CH1); if the codes are not identical, then the new code is accepted and stored in the key code FIFO (CH) and in the prior key code variable (CH1). If the codes are identical, then the new code is accepted only if a suitable key debounce delay has transpired since the prior value was accepted.

If the key code read and accepted is the code for CTRL-1, then the display start/stop flag (SSFLAG) is complemented and the value is not stored in the key code FIFO (CH).

In addition to the reading of the key data, SRTIMR is set to \$30 for all interrupts received (see E8), and ATRACT is set to 0 whenever a new code is accepted (see B10).

The Keyboard handler obtains all key data from CH; whenever a code is extracted from that one-byte FIFO, the handler stores a value of \$FF to the FIFO to indicate that the code has been read. See Chapter 5 for further discussion of the Keyboard handler's processing of the key codes.

E1 CH1* [02F2,1] -- Prior keyboard character code.

CH1 contains the key code value of the key most recently read and accepted.

E2 KEYDEL* [02F1,1] -- Debounce delay timer.

KEYDEL is set to a value of 3 whenever a key code is accepted, and is decremented every 60th of a second by the stage 2 VBLANK process (until it reaches zero).

E3 CH [02FC,1] -- Keyboard character code FIFO.

CH is a one-byte FIFO which contains either the value of the most recently read and accepted key code or the value \$FF (which indicates

that the FIFO is empty). The FIFO is normally read by the keyboard handler, but may be read by a user program.

Key data may also be stored into CH by the auto-repeat logic as explained in the discussion relating to EB.

Special functions

Start/stop

Display handler and Screen Editor output to the text or graphics mode screen may be stopped and started (without losing any of the output data) through the use of the CTRL-1 key combination. Each key depression toggles a flag which is monitored by the above mentioned handlers. When the flag is non-zero, the handlers wait for it to go to zero before continuing any output.

E4 SSFLAG [02FF,1] -- Start/stop flag

The flag is normally zero, indicating that screen output is not to be stopped. The flag is complemented by every occurrence of the CTRL-1 key combination by the keyboard IRQ service routine.

The flag is set to zero upon power up, [RESET] or [BREAK] key processing.

[BREAK] key

E5 BRKKEY [0011,1] -- [BREAK] key flag

BRKKEY is used to indicate that the [BREAK] key has been pressed. The value is normally non-zero and is set to zero whenever the [BREAK] key is pressed. The code that detects and processes the [BREAK] condition (flag = 0) should set the flag non-zero again.

BRKKEY is monitored by the following OS routines: Keyboard handler, Display handler, Screen Editor, Cassette handler, xx? The detection of a [BREAK] condition during an I/O operation, will cause the operation to be aborted and a status of \$80 to be returned to the user.

The flag is set non-zero upon power up, [RESET] or upon aborting a pending I/O operation.

SHIFT/CONTROL lock

The keyboard control has three different modes for code generation which apply to the alphabetic keys 'A' through 'Z': 1) normal, 2) caps lock and 3) control lock.

In normal mode, all unmodified alphabetic character keys generate the lower case letter ATASCII code (\$61-7A).

In caps lock mode, all unmodified alphabetic character keys generate the upper case letter ATASCII code (\$41-5A).

In control lock mode, all unmodified alphabetic character keys generate the control letter ATASCII code (\$01-1A).

In all three modes, any alphabetic character key which is modified (by being pressed in conjunction with the SHIFT or CONTROL key) will generate the desired modified code.

E6 SHFLOK [02BE,1] -- Shift/control lock control flag

SHFLOK normally has one of three values:

\$00 = normal mode (no locks in effect).
\$40 = caps lock.
\$80 = control lock.

SHFLOK is set to \$40 upon power up and [RESET] and is modified thereafter by the OS only when the CAPS key is pressed (either by itself or in conjunction with the SHIFT or CTRL key).

E7 HOLDCH* [007C,1] -- Character holding variable

HOLDCH is used to retain the current character value prior to the SHIFT/CONTROL logic process.

Auto-repeat

The auto-repeat feature responds to the continuous depression of a keyboard key by replicating the key code 10 times per second, after an initial 1/2 second delay. The timer variable SRTIMR is used to control both the initial delay and the repeat rate.

Whenever SRTIMR is equal to zero and a key is being held down, the value of the key code is stored in the key code FIFO (CH). This logic is part of the stage 2 VBLANK process.

E8 SRTIMR* [022B,1] -- auto-repeat timer

SRTIMR is controlled by two independent processes: 1) the keyboard IRQ service routine, which establishes the initial delay value and 2) the stage 2 VBLANK routine which establishes the repeat rate, decrements the timer and implements the auto-repeat logic.

Inverse video control

The Keyboard handler allows the direct generation of more than half of the 256 ATASCII codes; but codes \$80-\$9A and codes \$A0-\$FC can be generated only with the "inverse video mode" active. The ATARI key acts as an on/off toggle for this mode, and all characters (except for screen editing control characters) will be subject to inversion when the mode is active.

E9 INVFLG [02B6,1] -- inverse video flag

INVFLG is normally zero, indicating that normal video ATASCII codes (bit-7 = 0) are to be generated from keystrokes; however, whenever INVFLG is non-zero, inverse video ATASCII codes (bit-7 = 1) will be generated. The special control codes are exempt from this bit manipulation.

INVFLG is set to zero by power up and [RESET].

The Keyboard handler inverts bit-7 of INVFLG whenever the ATARI key is pressed; the lower order bits are not altered and are assumed to be zero.

The Keyboard handler "exclusive or"s the ATASCII key data with the value in INVFLG at all times; the normal values of \$00 and \$80 thus lead to control of the inverse video bit (bit-7).

Console switches (SELECT, START & OPTION)

The console switches are sensed directly from the hardware register CONSOL [D01F]; see the Colleen hardware manual for details.

F. PRINTER

See Chapter 5 for a general description of the Printer handler.

Printer buffer

F1 PRNBUF* [03C0,40] -- Printer record buffer

PRNBUF is the buffer used by the Printer handler for packing printer data to be sent to the device controller. The buffer is 40 bytes long and contains nothing but printer data.

F2 PBUFSZ* [001E,1] -- Printer record size

PBUFSZ contains the size of the printer record for the current mode selected; the modes and respective sizes (in decimal bytes) are shown below:

Normal	40
Double width	20 (not currently supported by the device)
Sideways	29

Status request 4

F3 PBPNT* [001D, 1] -- Printer buffer index

PBPNT contains the current index to the printer buffer. PBPNT ranges in value from zero to the value of PBUFSZ.

Internal working variables

F4 PTEMP* [001F, 1] -- Printer handler temporary data save

PTEMP is used by the Printer handler to temporarily save the value of a character to be output to the printer.

F5 PTIMOT* [001C, 1] -- Printer timeout value

PTIMOT contains the timeout value for SIO calling sequence variable DTIMLO (see Chapter 9); PTIMOT is set to 30 (which represents a 32 second timeout) at initialization time, and is updated after each printer status request operation to contain the value returned in the 3rd byte of the status frame (see Chapter 5).

G. CENTRAL I/O ROUTINE (CIO)

See Chapter 5 for a description of the Central I/O Utility.

User call parameters

CIO call parameters are passed primarily through an I/O Control Block (IOCB); although additional device status information may be returned in DVSTAT, and handler information is obtained from the Device Table (HATABS).

I/O Control Block

IOCB is the name applied collectively to the 16 bytes associated with each of the 8 provided control structures; see Chapter 5.

G1 IOCB [0340, 16] -- I/O Control Block

The label IOCB is the location of the first byte of the first IOCB in the data base. For VIDs G2 through G10, the addresses given are for IOCB #0 only, the addresses for all of the IOCBs are shown below:

0340-034F	IOPCB #0
0350-035F	IOPCB #1
0360-036F	IOPCB #2
0370-037F	IOPCB #3
0380-038F	IOPCB #4
0390-039F	IOPCB #5
03A0-03AF	IOPCB #6
03B0-03BF	IOPCB #7

G2 ICHID [0340, 1] -- Handler I. D.

See Chapter 5. Initialized to \$FF at power up and [RESET].

G3 ICDNO [0341, 1] -- Device number

See Chapter 5.

G4 ICCOM [0342, 1] -- Command byte

See Chapter 5.

G5 ICSTA [0343, 1] -- Status

See Chapter 5.

G6 ICBAL, ICBAH [0344, 2] -- Buffer address

See Chapter 5.

G7 ICPTL, ICPTH [0346, 2] -- PUT BYTE vector

See Chapter 5. Initialized to point to CIO's "IOPCB not OPEN" routine at power up and [RESET].

G8 ICBLL, ICBLH [0348, 2] -- Buffer length / byte count

See Chapter 5.

G9 ICAX1, ICAX2 [034A, 2] -- Auxilliary information

See Chapter 5.

G10 ICSPR [034C, 4] -- Spare bytes for handler use

There is no fixed assignment of these four bytes; the handler associated with an IOPCB may or may not use these bytes.

Device status

G11 DVSTAT [02EA, 4] -- Device status

See Chapter 5 for a discussion of the GET STATUS command.

Device Table

G12 HATABS [031A, 38] -- Device Table

See Chapter 9 for a description of the Device Table.

CIO/handler interface parameters

Communication between CIO and a handler is accomplished using the 6502 machine registers, and a data structure called the zero page IOCB (ZIOCB). The ZIOCB is essentially a copy of the particular IOCB being used for the current operation.

Zero page IOCB

G13 ZIOCB (IOCBAS) [0020, 16] -- Zero page IOCB

The zero page IOCB is an exact copy (except as noted in the discussions that follow) of the IOCB specified by the 6502 X register upon entry to CIO; CIO copies the outer level IOCB to the zero page IOCB, performs the indicated function, moves the (possibly altered) zero page IOCB back to the outer level IOCB, and then returns to the caller.

Although both the outer level IOCB and the zero page IOCB are defined to be 16 bytes in size, only the first 12 bytes are moved by CIO.

G14 ICHIDZ [0020, 1] -- Handler index number

See Chapter 5. Set to \$FF on CLOSE.

G15 ICDNOZ [0021, 1] -- Device drive number

See Chapter 5.

G16 ICCOMZ [0022, 1] -- Command byte

See Chapter 5.

G17 ICSTAZ [0023, 1] -- Status byte

See Chapter 5.

G18 ICBALZ, ICBALH [0024, 2] -- Buffer address

See Chapter 5. This pointer variable is modified by CIO in the course of processing some commands; however, the original value is restored before returning to the caller.

G19 ICPTLZ, ICPTHZ

See Chapter 5. Set to point to CIO's "IOCB not OPEN" routine on CLOSE.

G20 ICBLLZ, ICBLHZ [0028, 2] -- Buffer length / byte count

See Chapter 5. This double byte variable, which starts out representing the buffer length, is modified by CIO in the course of processing some commands; then, before returning to the caller, the transaction byte count is stored therein.

G21 ICAX1Z, ICAX2Z [002A, 2] -- Auxilliary information

See Chapter 5.

G22 ICSPRZ (ICIDNO, CIOCHR) [002C, 4] -- CIO working variables

ICSPRZ and ICSPRZ+1 are used by CIO in obtaining the appropriate handler entry point from the handler's vector table (see Chapter 9).

ICSPRZ+2 is also labeled ICIDNO and retains the value of the 6502 X register from CIO entry. The X register is loaded from ICIDNO as CIO returns to the caller.

ICSPRZ+3 is also labeled CIOCHR and retains the value of the 6502 A register from CIO entry, except for data reading type commands, in which case the most recent data byte read is stored in CIOCHR. The 6502 A register is loaded from CIOCHR as CIO returns to the caller.

Internal working variables

G23 ICCOMT* [0017, 1] -- Command table index

ICCOMT is used as an index to CIO's internal command table, which maps command byte values to handler entry offsets (see Chapter 9 for more information). ICCOMT contains the value from ICCOMZ except when ICCOMZ is greater than \$OE, in which case ICCOMT is set to \$OE.

G24 ICIDNO* [002E, 1] -- CIO call X register save/restore

See G22.

G25 CIOCHR* [002F, 1] -- CIO call A register save/restore

See G22.

H. SERIAL I/O ROUTINE (SIO)

See Chapter 9 for discussions relating to SIO.

User call parameters

SIO call parameters are passed primarily through a Device Control Block; although an additional "noisy bus" option exists which is selectable through a separate variable.

Device Control Block

H1 DCB [0300, 12] -- Device Control Block

DCB is the name applied collectively to the 12 bytes at locations 0300-030B. These bytes provide the parameter passing mechanism for SIO and are described individually below.

H2 DDEVIC [0300, 1] -- Device bus I.D.

See Chapter 9.

H3 DUNIT [0301, 1] -- Device unit number

See Chapter 9.

H4 DCOMMAND [0302, 1] -- Device command

See Chapter 9.

H5 DSTATUS [0303, 1] -- Device status

See Chapter 9.

H6 DBUFLO, DBUFHI [0304, 2] -- Handler buffer address

See Chapter 9.

H7 DTIMLO [0306, 1] -- Device timeout

See Chapter 9.

H8 DBYTLO, DBYTHI [0308, 2] -- Buffer length / byte count

See Chapter 9.

H9 DAUX1, DAUX2 [030A, 2] -- Auxilliary information

See Chapter 9.

Bus sound control

H10 SOUNDR [0041, 1] -- Quiet/noisy I/O flag

SOUNDR is a flag used to indicate to SIO whether noise is to be generated on the TV audio circuit when Serial I/O bus activity is in progress. SOUNDR equal to zero indicates that sound is to be inhibited, and non-zero indicates that sound is to be enabled. SIO sets SOUNDR to 3 at power up and [RESET].

Serial bus control

Retry logic

SIO will attempt one complete command retry if the first attempt is not error free, where a complete command try consists of up to 14 attempts to send (and acknowledge) a command frame, followed by a single attempt to receive COMPLETE and possibly a data frame.

H11 CRETRY* [0036,1] -- Command frame retry counter

CRETRY controls the inner loop of the retry logic, that associated with sending and receiving an acknowledgement of the command frame. CRETRY is set to 13 by SIO at the beginning of every command initiation, thus allowing for an initial attempt and up to 13 additional retries..

H12 DRETRY* [0037,1] -- Device retry counter

DRETRY controls the outer loop of the retry logic, that associated with initiating a command retry after a failure subsequent to the command frame acknowledgement. DRETRY is set to 1 by SIO at entry, thus allowing for an initial attempt and up to 1 additional retry.

Checksum

The Serial I/O bus protocol specifies that all command and data frames must contain a checksum validation byte; this byte is the arithmetic sum (with end-around carry) of all of the other bytes in the frame.

H13 CHKSUM* [0031,1] -- Checksum value

CHKSUM contains the frame checksum as computed by SIO for all frame transfers.

H14 CHKSNT* [003B,1] -- Checksum sent flag

CHKSNT indicates to the serial bus transmit interrupt service routine whether the frame checksum byte has been sent yet. CHKSNT equal to zero indicates that the checksum byte has not yet been sent; after the checksum is sent, CHKSNT is then set non-zero.

H15 NOCKSM* [003C,1] -- No checksum follows data flag

NOCKSM is a flag used to communicate between the SIO top level code and the Serial bus receive interrupt service routine that the next input will not be followed by a checksum byte. A value of zero specifies that a checksum byte will follow, non-zero that a checksum byte will not follow.

Data buffering

General buffer control

H16 BUFRLO* [0032,1] & BUFRHI* [0033,1] -- Next byte address

BUFRLO and BUFRHI comprise a pointer to the next buffer location to be read from or written to. For a data frame transfer, the pointer is initially set to the value contained in the SIO call parameters DBUFL0 and DBUFHI, and is then incremented by the interrupt service routines as a part of normal bus data transfer. For a command frame transfer, the pointer is set to point to the SIO maintained command frame output buffer.

H17 BFENLO* [0034,1] & BFENHI* [0035,1] -- Buffer end address

BFENLO/BFENHI form a pointer to the byte following the last frame data byte (not including the checksum) to be sent or received. BFENLO/BFENHI is the arithmetic sum of BUFRLO/BUFRHI plus the frame size plus -1.

Command frame output buffer

See Chapter 9 for the command frame format and description.

H18 CDEVIC* [023A,1] -- Command frame device I.D.

CDEVIC is set to the value obtained by adding SIO call parameter DDEVIC to DUNIT and subtracting one.

H19 CCOMMAND* [023B,1] -- Command frame command.

CCOMND is set to the value obtained from SIO call parameter DCOMND.

H20 CAUX1* [023C,1] & CAUX2* [023D,1] -- Auxilliary info

CAUX1 and CAUX2 are set to the values obtained from SIO call parameters DAUX1 and DAUX2, respectively.

Receive/transmit data buffering

H21 BUFRFL* [0038,1] -- Buffer full flag

BUFRFL is a flag used by the serial bus receive interrupt service routine to indicate when the main portion of a bus frame has been received -- all but the checksum byte. BUFRFL equal to zero indicates that the main portion has not been completely received, a non-zero value indicates that the main portion has been received.

H22 RECVDN* [0039,1] -- Receive frame done flag

RECVDN is a flag used by SIO to communicate between the Serial bus receive interrupt service routine and the main SIO code. The flag is initially set to zero by SIO, and later set non-zero by the interrupt service routine after the last byte of a bus frame has been received.

H23 TEMP* [023E,1] -- SIO one-byte I/O data

TEMP is used to receive one-byte responses from serial bus controllers, such as ACK, NAK, COMPLETE or ERROR.

H24 XMTDON* [003A,1] -- Transmit frame done flag

XMTDON is a flag used by SIO to communicate between the Serial bus transmit interrupt service routine and the main SIO code. The flag is initially set to zero by SIO, and later set non-zero by the interrupt service routine after the last byte of a bus frame has been transmitted.

SIO timeout

SIO uses system timer 1 to provide the timeout capability for various operations initiated internally. See Chapter 6 for a discussion of the capabilities of the system timers. TIMFLG is the flag used to communicate between SIO and the timer initiated code pointed to by CDTMA1.

H25 TIMFLG* [0317,1] -- SIO operation timeout flag

TIMFLG is used to indicate a timeout situation for a bus operation. The flag is initially set to one, and if it attains a value of zero (after the timeout period) before the current operation is complete, the operation will be aborted. See also D2.

H26 CDTMV1* [0218,2] -- System timer 1 value

This two-byte count takes on various values depending upon the operation being timed. See also P4.

H27 CDTMA1* [0226,2] -- System timer 1 address

This vector always points to the 'JTIMER' routine, whose only function is to set TIMFLG to zero. This vector is initialized by SIO before every use, so that system timer 1 may be used by any process that does not use SIO within a timing function. See also P5.

Internal working variables

H28 STACKP* [0318,1] -- Stack pointer save/restore

STACKP contains the value of the 6502 SP register at entry to SIO; this is retained to facilitate a direct error exit from an SIO

subroutine.

H29 TSTAT* [0319, 1] -- Temporary status

TSTAT is used to return the operation status from the 'WAIT' routine and will contain one of the SIO status byte values as shown in Appendix B.

H30 ERRFLG* [023F, 1] -- I/O error flag

ERRFLG is used for communication between the 'WAIT' routine and the outer level SIO code. ERRFLG is normally zero, but is set to \$FF when a device responds with an invalid response byte.

H31 STATUS* [0030, 1] -- SIO operation status

STATUS is a zero page variable that is used within SIO to contain the operation status that will be stored to the calling sequence parameter variable DSTATUS when SIO returns to the caller.

H32 SSKCTL* [0232, 1] -- SKCTL copy

SSKCTL is utilized by SIO to keep track of the content of the SKCTL [D20F] register which is a write only register.

J. ATARI CONTROLLERS

Various of the ATARI controllers are read as part of the Stage 2 VBLANK process; the encoded data is partially decoded and processed as shown in the sub-sections that follow.

Joysticks

Up to 4 joystick controllers may be attached to the computer console, each with a 9 position joystick plus a trigger button.

J1 STICK0 - STICK3 [0278, 4] -- Joystick position sense

The four joystick position sense variables contain a bit encoded position sense as shown below:

7	6	5	4	3	2	1	0	
+	-	+	-	+	-	+	-	
1	0	0	0	!R!	I	L!	D!	U!
+	-	+	-	+	-	+	-	

where: R = 0 indicates joystick RIGHT sensor true.
L = 0 indicates joystick LEFT sensor true.

D = 0 indicates joystick DOWN sensor true.
U = 0 indicates joystick UP sensor true.

Nine unique combinations are possible, indicating the possible joystick positions shown below:

CENTER	\$0F
UP	\$0E
UP/RIGHT	\$06
RIGHT	\$07
DOWN/RIGHT	\$05
DOWN	\$0D
DOWN/LEFT	\$09
LEFT	\$0B
UP/LEFT	\$0A

J2 STRIGO - STRIG3 [0284, 4] -- Joystick trigger sense

The four joystick trigger sense variables each contain a single bit indicating the position of the joystick trigger as shown below:

7 6 5 4 3 2 1 0
+-+-+---+-+---+
|0 0 0 0 0 0 |T:
+-+-+---+-+---+

where: T = 0 indicates trigger pressed.

Paddles

Up to eight paddle controllers may be connected to the computer, each with a potentiometer and a trigger sense.

J3 PADDL0 - PADDL7 [0270, 8] -- Paddle position sense

There is a single byte variable associated with each paddle position sense; the values range from 228 for full counterclockwise rotation to 1 for full clockwise rotation.

The paddle values are often converted by the user, as shown below, to give a result of 0 for full counterclockwise rotation and 227 for full clockwise rotation:

VALUE := 228 - PADDLX;

J4 PTRIGO - PTRIG7 [027C, 8] -- Paddle trigger sense

The eight paddle trigger sense variables each contain a single bit indicating the position of the paddle trigger as shown below:

```
7 6 5 4 3 2 1 0
+-+-+-+---+---+
!0 0 0 0 0 0 !T!
+-+-+-+---+---+
```

where: T = 0 indicates trigger pressed.

Light pen

The OS reads the position of a single light pen and stores the horizontal and vertical position codes in two variables; these codes are not the same as the actual screen coordinates. The pen position codes for different portions of the screen are shown below:

Left edge -- 67.

Codes increase monotonically to a value of 227, then go to 0 and continue to increase monotonically (one count per color clock).

Right edge -- 7.

Upper edge -- 16.

Codes increase monotonically (one count per two raster lines).

Lower edge -- 111.

The light pen hardware will read and latch the pen position 60 times per second, independent of the pen button position, which is separately sensed.

In order for the light pen to operate it must be positioned over a portion of the screen which has sufficient luminance to activate the photosensor in the pen; a blank (dark) screen will generally not provide enough luminance to utilize the light pen.

J5 LPENH [0234, 1] -- Light pen horizontal position code

LPENH contains the horizontal position code for the light pen; the algorithm below (written in Pascal) shows the conversion from position code to screen coordinate (screen mode 7):

```
IF LPENH < 33      { check for rollover point }
  THEN            { adjust values to right of rollover }
    XPOS := LPENH + 227
  ELSE            { no adjustment to left of rollover point }
    XPOS := LPENH;
XPOS := XPOS - 67; { adjust for left edge offset }
IF XPOS < 0 THEN XPOS := 0;
IF XPOS > 159 THEN XPOS := 159;
```

J6 LPENV [0235, 1] -- Light pen vertical position code

LPENV contains the vertical position code for the light pen; the algorithm below (written in Pascal) shows the conversion from position code to screen coordinate (screen mode 7):

```
YPOS := LPENV - 16; { adjust for upper edge offset }
IF YPOS < 0 THEN YPOS := 0;
IF YPOS > 95 THEN YPOS := 95;
```

J7 STICKO - STICK3 [0278, 4] -- Lightpen button sense

The lightpen button sense is encoded in one of STICKO - STICK3 (depending upon the actual controller port used) as shown below:

7	0
+-----+	
1 1 1 1 T	
+-----+	

where: T = 0 indicates the lightpen button is pressed.

The lightpen button sense is encoded in one of STRICKO - STRICK3 [0284, 4] as shown below:

7	0
+-----+	
0 0 0 0 0 0 T	
+-----+	

Driving controllers

The driving controller has no position stops and thus allows unlimited rotation in either direction; the output of the controller is a 2-bit Gray code which can be used to determine the direction of rotation. The controller is sensed using the same internal hardware as the joystick, thus the same data base variables are used for both.

J8 STICKO - STICK3 [0278, 4] -- Driving controller sense

The four driving controller sense variables contain an encoded rotation (position) sense value, as shown below:

7 6 5 4 3 2 1 0							
+-----+							
0 0 0 1 1 val							
+-----+							

where a clockwise rotation of the controller produces the following continuous sequence of four values (shown in hexadecimal):

OF, OD, OC, OE, OF, OD,

and a counterclockwise rotation of the controller produces the following continuous sequence of four values:

OF, OE, OC, OD, OF, OE,

J9 STRIGO - STRIG3 [0284, 4] -- Driving trigger sense

The four driving trigger sense variables each contain a single bit indicating the position of the driving trigger as shown below:

7	6	5	4	3	2	1	0
+	+	+	+	+	+	+	+
!0	0	0	0	0	0	!	T!
+	+	+	+	+	+	+	+

where: T = 0 indicates trigger pressed.

K. DISK FILE MANAGER

See Chapter 5 for information relating to the Disk File Manager.

K1 FMSZPG* [0043, 7] -- FMS reserved space

FMSZPG is the reserved space in the database for the variables shown below; the names associated with K2 through K5 are not in the system equate file.

K2 ZBUFP* [0043, 2] -- Buffer pointer

K3 ZDRVAP* [0045, 2] -- Drive pointer

K4 ZSBA* [0047, 2] -- Sector buffer pointer

K5 ERRNO* [0049, 1] -- Error number

L1 DSKUTL* [001A, 2] -- Page zero pointer variable

M. FLOATING POINT PACKAGE

See Chapter 8 for a description of the Floating Point Package.

- M1 FRO [00D4,6] -- F.P. register 0
 - M2 FRE* [00DA,6] -- F.P. register (internal)
 - M3 FR1 [00E0,6] -- F.P. register 1
 - M4 FR2* [00E6,6] -- F.P. register 2 (internal)
 - M5 FRX* [00EC,1] -- Spare (unused)
 - M6 EEXP* [00ED,1] -- Exponent value (internal)
 - M7 NSIGN* [00EE,1] -- Sign of mantissa (internal)
 - M8 ESIGN* [00EF,1] -- Sign of exponent (internal)
 - M9 FCHRFLG* [00F0,1] -- First character flag (internal)
 - M10 DIGRT* [00F1,1] -- Digits to right of decimal point
 - M11 CIX [00F2,1] -- Character index
 - M12 INBUFF [00F3,2] -- Input text buffer pointer
 - M13 ZTEMP1* [00F5,2] -- Temporary storage
 - M14 ZTEMP4* [00F7,2] -- Temporary storage
 - M15 ZTEMP3* [00F9,2] -- Temporary storage
 - M16 FLPTR [00FC,2] -- Pointer to F.P. number
 - M17 FPTR2* [00FE,2] --
 - M18 LBPR1* [057E,1] -- LBUFF preamble
 - M19 LBPR2* [057F,1] -- LBUFF preamble
 - M20 LBUFF [0580,96] -- Text buffer
 - M21 PLYARG* [05E0,6] -- F.P. register (internal)
 - M22 FPSCR/FSCR* [05E6,6] -- F.P. register (internal)
 - M23 FPSCR1/SCR1* [05EC,6] -- F.P. register (internal)
 - M24 DEGFLG/RADFLG [00FB,1] -- Degrees/radians flag
- DEGFLG = 0 indicates radians, 6 indicates degrees.

N. POWER UP & [RESET]

See Chapter 7 for details of the power up and [RESET] operations.

RAM sizing

During power up and [S RESET] the first non-RAM address above 1000 hex is located and its address retained using a non-destructive test. The first byte of every 4K memory "block" is tested to see if it is alterable; if so, the original value is restored and the next block is tested, and if not, that address is considered to be the end of RAM.

N1 RAMLO*/TRAMSZ* [0004,3] -- RAM data/test pointer (temporary)

RAMLO+1 contains the lsb of the address to be tested (always = 0) and TRAMSZ (same as RAMLO+2) contains the msb of the address to be tested. RAMLO+0 contains the complemented value of the data originally contained in the memory location being tested.

Later in the initialization process these variables are used for totally unrelated functions; but first the value in TRAMSZ is moved to the variables RAMSIZ and MEMTOP+1.

N2 TSTDAT* [0007,1] -- Test data byte save

TSTDAT contains the original value of the memory location being tested.

Disk/cassette boot

As a part of the power up sequence, software may be booted from an attached disk drive or cassette player as explained in Chapter 10.

N3 DOSINI [000C,2] -- Disk boot initialization vector.

DOSINI contains the disk booted software initialization address from the beginning of the boot file (See Chapter 10) whenever a disk boot is successfully completed.

N4 CKEY* [004A,1] -- Cassette boot request flag

CKEY is an internal flag used to indicate that the console START key was pressed during power up, thus indicating that a cassette boot is desired. CKEY equals zero when no cassette boot is requested, and is non-zero when a cassette boot is requested. The flag is cleared to zero after a cassette boot.

N5 CASSBT* [004B,1] -- Cassette booting flag

CASSBT is used during the cassette boot process to indicate to shared code that the cassette is being booted and not the disk. CASSBT equal to zero indicates a disk boot, and non-zero indicates a cassette boot.

N6 CASINI [0002,2] -- Cassette boot initialization vector

CASINI contains the cassette booted software initialization address from the beginning of the boot file (See Chapter 10) whenever a cassette boot is successfully completed.

N7 BOOT?* [0009,1] -- Successful disk/cassette boot flag.

BOOT? indicates to the initialization processor which, if any, of the boot operations went to successful completion. The flag values are set by the OS and the format for the variable is shown below:

```
 7 6 5 4 3 2 1 0
+-+-+-+---+---+
 |           |C|D|
 +-----+---+---+
```

where: C = 1 indicates that the cassette boot went to completion.
D = 1 indicates that the disk boot went to completion.

N8 DFLAGS* [0240,1] -- Disk flags

DFLAGS contains the value of the first byte of the boot file, after a disk boot. See Chapter 10.

N9 DBSECT* [0241,1] -- Disk boot sector count

DBSECT is initially set to the value of the second byte of the boot file, during a disk boot, and is then used to control the number of additional disk sectors read, if any.

N10 BOOTAD* [0242,2] -- Disk boot memory address

BOOTAD is initially set to the value of the 3rd and 4th bytes of the boot file, during a disk boot, and is not modified thereafter.

Environment control

If, at the end of a power up or [RESET], control is not given to one of the cartridges (as explained in sections 7 and 10), then program control passes to the address contained in the data base variable DOSVEC.

N11 COLDST* [0244,1] -- Coldstart complete flag

COLDST is used by the initialization routine to detect the case of a [RESET] occurring before the completion of the power up process. COLDST is set to \$FF at the beginning of the power up sequence and is set to 0 at the completion; if a [RESET] occurs while the value is non-zero, the power up sequence will be reinitiated (rather than initiating a [RESET] sequence).

N12 DOSVEC [000A,2] -- Non-cartridge control vector

At the beginning of power up the OS sets DOSVEC to point to the "blackboard" routine; DOSVEC may then be altered as a consequence of a disk boot or cassette boot (as explained in Chapter 10) to establish a new control program. Control will be passed through DOSVEC on all power up and [S RESET] conditions in which a cartridge does not take control first.

[S RESET]

N13 WARMST [0008, 1] -- Warmstart flag

WARMST equals \$FF during a [RESET] (warmstart) initialization and equals 0 during a power up initialization (coldstart).

P. INTERRUPTS

See Chapter 6 for a discussion of interrupt processing.

P1 CRITIC [0042, 1] -- Critical code section flag

CRITIC is used to signal to the VBLANK interrupt processor that a critical code section is executing without IRQ interrupts being inhibited; the VBLANK interrupt processor will stop interrupt processing after stage 1 and before stage 2, just as if the 6502 processor I bit were set, when CRITIC is set.

CRITIC equal to zero indicates that the currently executing code section is non-critical, while any non-zero value indicates that the currently executing code section is critical.

P2 POKMSK [0010, 1] -- POKEY interrupt mask

POKMSK is a software maintained interrupt mask that is used in conjunction with the enabling and disabling of the various POKEY interrupts. This mask is required because the POKEY interrupt enable register IRQEN [D2OE] is a write only register, and at any point in time the system may have several users independently enabling and disabling POKEY interrupts. POKMSK is updated by the users to always contain the current content of IRQEN.

System timers

The system timers are discussed in detail in Chapter 6.

Real time clock

The real time clock (or frame counter, as it is sometimes called) is incremented as part of the stage 1 VBLANK process as explained in Chapter 6.

P3 RTCLOK [0012,3] -- Real time frame counter

RTCLOK+0 is the most significant byte, RTCLOK+1 the next most significant byte, and RTCLOK+2 the least significant byte. See the discussions at D3 and preceding B10 for OS use of RTCLOK.

System timer 1

System timer 1 is maintained as part of the stage 1 VBLANK process, and thus has the highest priority of any of the user timers.

P4 CDTMV1 [0218,2] -- System timer 1 value

CDTMV1 contains zero when the timer is inactive, otherwise it contains the number of VBLANKs remaining until timeout. Also see H26

P5 CDTMA1 [0226,2] -- System timer 1 jump address

CDTMA1 contains the address to which to JSR should the timer timeout. See also H27 and Chapter 6.

System timer 2

System timer 2 is maintained as part of the stage 2 VBLANK process, and has the second highest priority of the user timers. The OS does not have any direct use for system timer 2.

P6 CDTMV2 [021A,2] -- System timer 2 value

CDTMV2 contains zero when the timer is inactive, otherwise it contains the number of VBLANKs remaining until timeout.

P7 CDTMA2 [0228,2] -- System timer 2 jump address

CDTMA2 contains the address to which to JSR should the timer timeout. See Chapter 6.

System timers 3, 4 and 5

System timers 3, 4 and 5 are maintained as part of the stage 2 VBLANK process, and have the lowest priority of the user timers. The OS does not have any direct use for these timers.

P8 CDTMV3 [021C,2], CDTMV4 [021E,2] & CDTMV5 [0220,2]

These variables contain zero when the corresponding timers are inactive, otherwise they contain the number of VBLANKs remaining until timeout.

P9 CDTMF3 [022A, 1], CDTMF4 [022C, 1] & CDTMF5 [022E, 2]

Each of these one-byte variables will be set to zero should its corresponding timer timeout. The OS never modifies these bytes except to set them to zero upon timeout (and initialization).

RAM interrupt vectors

There are RAM vectors for many of the interrupt conditions within the system. See Chapter 6 for a discussion of the placing of values to these vectors.

NMI interrupt vectors

P10 VDSLST [0200, 2] -- Display list interrupt vector

This vector is not used by the OS. See Chapters 6.

P11 VVBLKI [0222, 2] -- Immediate VBLANK vector

This vector is initialized to point to the OS stage 1 VBLANK

P12 VVBLKD [0224, 2] -- Deferred VBLANK vector

This vector is initialized to point to the OS VBLANK exit routine. See Chapter 6.

IRQ interrupt vectors

P13 VIMIRQ [0216, 2] -- General IRQ vector

This vector is initialized to point to the OS IRQ interrupt processor. See Chapter 6.

P14 VPRCED [0202, 2] -- Serial I/O bus proceed signal

The serial bus line that produces this interrupt is not used in the current system. See Chapter 6.

P15 VINTER [0204, 2] -- Serial I/O bus interrupt signal

The serial bus line that produces this interrupt is not used in the current system. See Chapter 6.

P16 V[BREAK] [0206, 2] -- BRK instruction vector

This vector is initialized to point to a PLA, RTI sequence as the OS proper does not utilize the BRK instruction. See Chapter 6.

P17 VKEYBD [0208,2] -- Keyboard interrupt vector

This vector is initialized to point to the Keyboard handler's interrupt service routine. See Chapter 6 and the discussion preceding E1.

P18 VSERIN [020A,2] -- Serial I/O bus receive data ready

This vector is initialized to point to the SIO utility's interrupt service routine. See Chapter 6.

P19 VSEROR [020C,2] -- Serial I/O bus transmit ready

This vector is initialized to point to the SIO utility's interrupt service routine. See Chapter 6.

P20 VSEROC [020E,2] -- Serial I/O bus transmit complete

This vector is initialized to point to the SIO utility's interrupt service routine. See Chapters 6 and 6.

P21 VTIMR1 [0210,2], VTIMR2 [0212,2] & VTIMR4 [0214,2] -- POKEY timer vectors

The POKEY timer interrupts are not used by the OS See Chapter 6.

Hardware register updates

As part of the stage 2 VBLANK process, certain hardware registers are updated from OS database variables as explained in Chapter 6.

P22 SDMCTL* [022F,1] -- DMA control

SDMCTL is set to a value of \$02 at the beginning of a Display handler OPEN command, and then later set to a value of \$22. The value of SDMCTL is stored to DMACTL [D400] as part of the stage 2 VBLANK process.

P23 SDLSTL* [0230,1] & SDLSTH* [0231,1] -- Display list address

The Display handler formats a new display list with every OPEN command and puts the display list address in SDLSTL and SDLSTH. The value of these bytes are stored to DLISTL [D402] and DLISTH [D403] as part of the stage 2 VBLANK process.

0360-036F	IOCB #2
0370-037F	IOCB #3
0380-038F	IOCB #4
0390-039F	IOCB #5

03A0-03AF	IOCB #6
03B0-03BF	IOCB #7

NOTE: There is a potential timing problem associated with the updating of the hardware registers from the database variables. Since the stage 2 VBLANK process is performed with interrupts enabled, it is possible for an IRQ interrupt to occur before the updating of DLISTH and DLISTL. If the processing of that interrupt (plus other nested interrupts) exceeds the vertical blank delay (1 msec.) then the display list pointer register will not have been updated when display list processing commences for the new frame, and a screen glitch will result.

P24 GPRIOR* [026F, 1] -- Priority control

The Display handler alters bits 6 and 7 of GPRIOR as part of establishing the GTIA mode. The value of GPRIOR is stored to PRIOR [D01B] as part of the stage 2 VBLANK process.

P25 CHACT* [02F3, 1] -- Character control

The Display handler sets CHACT to \$02 on every OPEN command. The value of CHACT is stored to CHACTL [D401] as part of the stage 2 VBLANK process.

P26 CHBAS [02F4, 1] -- Character address base

The Display handler sets CHBAS to \$E0 on every OPEN command. The value of CHBAS is stored to CHBASE [D409] as part of the stage 2 VBLANK process. This variable controls the character subset for screen modes 1 and 2; a value of \$E0 provides the capital letters and number set whereas a value of \$E2 provides the lower case letters and special graphics set. See B55 for more information.

P27 PCOLRx [02C0, 4] & COLORx [02C4, 5] -- Color registers

See B7 and B8.

Internal working variables

P28 INTEMP* [022D, 1] -- Temporary storage

INTEMP is used by the SETVBL (SETVBV) routine.

R. USER AREAS

The areas shown below are available to the user in a non-nested environment; See Chapter 4 for further information.

R1 [0080, 128]

R2 [0480, 640]

S. UNUSED (SPARE) BYTES

Labeled bytes

The labeled bytes listed below are thought to: 1) have no function within the OS and 2) not be modified except at initialization time.

S1 HOLD5* [02BD, 1]

S2 CSTAT* [0288, 1]

S3 DUNUSE* [0307, 1]

S4 TEMP2* [0314, 1]

S5 TMPX1* [029C, 1]

S6 DSKFMS* [0018, 2]

Unlabeled bytes

The unlabeled bytes and regions listed below are thought to: 1) have no function within the OS and 2) not be modified except at initialization time.

S7 [0000, 2] Reserved for LNBUG.

S8 [0236, 4]

S9 [0245, 1]

S10 [028B, 5]

S11 [02C9, 27] Reserved for LNBUG

S12 [02F5, 5]

S13 [03E8, 21]

S14 [0233, 1] Reserved for LNBUG

S15 [02E9, 1] Reserved for LNBUG

ALPHABETICAL LIST OF DATABASE VARIABLES

NAME	VID	ADDRESS SIZE
ADDCOR	D4	030E, 1
ADRESS	B39	0064, 2
APPMHI	A3	000E, 2
ATACHR	B54	02FB, 1
ATRACT	B10	004D, 1
BFENHI	H17	0035, 1
BFENLO	H17	0034, 1
BITMSK	B50	006E, 1
BLIM	D10	028A, 1
BOOT?	N7	0009, 1
BOOTAD	N10	0242, 2
BOTSCR	B16	02BF, 1
BPTR	D11	003D, 1
BRKKEY	E5	0011, 1
BUFADR	C1	0015, 2
BUFCNT	B42	006B, 1
BUFRFL	H21	0038, 1
BUFRHI	H16	0033, 1
BUFRLO	H16	0032, 1
BUFSTR	B43	006C, 2
CASBUF	D9	03FD, 131
CASFQ	D8	030F, 1
CASINI	N6	0002, 2
CASSBT	N5	004B, 1
CAUX1	H20	023C, 1
CAUX2	H20	023D, 1
CBAUDH	D1	02EF, 1
CBAUDL	D1	02EE, 1
CCOMND	H19	023B, 1
CDEVIC	H18	023A, 1
CDTMA1	P5, H27	0226, 2
CDTMA2	P7	0228, 2
CDTMF3	P9	022A, 1
CDTMF4	P9	022C, 1
CDTMF5	P9	022E, 1
CDTMV1	P4, H26	0226, 2
CDTMV2	P6	021A, 2
CDTMV3	P8	021C, 2
CDTMV4	P8	021E, 2

CDTMVS	P8	0220, 2
CH	E3	02FC, 1
CHKSNT	H14	003B, 1
CH1	E1	02F2, 1
CHACT	P25	02F3, 1
CHAR	B55	02FA, 1
CHBAS	P26	02F4, 1
CHKSNT	H14	003B, 1
CHKSUM	H13	0031, 1
CIOCHR	G25	002F, 1
CIX	M11	00F2, 1
CKEY	N4	004A, 1
COLAC	B24	0072, 2
COLCRS	B2	0055, 2
COLDST	N11	0244, 1
COLINC	B21	007A, 1
COLORO	B8, P27	02C4, 1
COLOR1	B8, P27	02C5, 1
COLOR2	B8, P27	02C6, 1
COLOR3	B8, P27	02C7, 1
COLOR4	B8, P27	02C8, 1
COLRSH	B11	004F, 1
COUNTR	B23	007E, 2
CRETRY	H11	0036, 1
CRITIC	P1	0042, 1
CRSINH	B1	02F0, 1
CSTAT	S2	0288, 1
DAUX1	H9	030A, 1
DAUX2	H9	030B, 2
DBSECT	N9	0241, 1
DBUFHI	H6	0304, 1
DBUFLO	H6	0305, 1
DBYTHI	H8	0308, 1
DBYTLO	H8	0309, 1
DCB	H1	0300, 12
DCOMND	H4	0302, 1
DDEVIC	H2	0300, 1
DEGFLG	M24	00FB, 1
DELTAC	B22	0077, 2
DELTAR	B22	0076, 1
DFLAGS	N8	0240, 1
DIGRT	M10	00F1, 1
DINDEX	B35	0057, 1
DMASK	B28	02A0, 1
DOSINI	N3	000C, 2
DOSVEC	N12	000A, 2
DRETRY	H12	0037, 1
DRKMSK	B12	004E, 1
DSKFMS	S6	0018, 2
DSKTIM	C2	0246, 1
DSKUTL	L1	001A, 2
DSPFLG	B27	02FE, 1
DSTAT	B34	004C, 1

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555 - Rev. 0.2

DSTATS	H5	0303, 1
DTIMLO	H7	0306, 1
DUNIT	H3	0301, 1
DUNUSE	S3	0307, 1
DVSTAT	G11	02EA, 4
EEXP	M6	00ED, 1
ENDPT	B25	0074, 2
ERRFLG	H30	023F, 1
(ERRNO	K5)	0049, 1
ESCFLG	B26	02A2, 1
ESIGN	M8	00EF, 1
FCHRFL	M9	00F0, 1
FEOF	D12	003F, 1
FILDAT	B17	02FD, 1
FILFLG	B18	02B7, 1
FLPTR	M16	00FC, 2
FMSZPG	K1	0043, 7
FPSCR	M22	05E6, 6
FPSCR1	M23	05EC, 6
FPTTR2	M17	00FE, 2
FRO	M1	00D4, 6
FR1	M3	00E0, 6
FR2	M4	00E6, 6
FRE	M2	00DA, 6
FREQ	D15	0040, 1
FRMADR	B41	0068, 2
FRX	M5	00EC, 1
FSCR	M22	05E6, 6
FSCR1	M23	05EC, 6
FTYPE	D13	003E, 1
GPRIOR	P24	026F, 1
HATABS	G12	031A, 38
HOLD1	B30	0051, 1
HOLD2	B31	029F, 1
HOLD3	B32	029D, 1
HOLD4	B20	02BC, 1
HOLD5	S1	02BD, 1
HOLDCH	E7	007C, 1
ICAX1	G9	034A, 1
ICAX1Z	G21	002A, 1
ICAX2	G9	034B, 1
ICAX2Z	G21	002B, 1
ICBAH	G6	0345, 1
ICBAHZ	G18	0025, 1
ICBAL	G6	0344, 1
ICBALZ	G18	0024, 1
ICBLH	G8	0349, 1
ICBLHZ	G20	0029, 1
ICBLL	G8	0348, 1

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555 - Rev. 0.2

ICBLLZ	G20	0028, 1
ICCOM	G4	0342, 1
ICCOMT	G23	0017, 1
ICCOMZ	G16	0022, 1
ICDNO	G3	0341, 1
ICDNOZ	G15	0021, 1
ICHID	G2	0340, 1
ICHIDZ	G14	0020, 1
ICIDND	G24, G2	2002E, 1
ICPTH	G7	0347, 1
ICPTHZ	G19	0027, 1
ICPTL	G7	0346, 1
ICPTLZ	G19	0026, 1
ICSPR	G10	034C, 4
ICSPRZ	G22	002C, 4
ICSTA	G5	0343, 1
ICSTAZ	G17	0023, 1
INBUFF	M12	00F3, 2
INSDAT	B45	007D, 1
INTEMP	P28	022D, 1
INVFLG	E9	02B6, 1
IOCB	G1	0340, 16
IOCBAS	G13	0020, 16
KEYDEL	E2	02F1, 1
LBFEND	M20	0580, 96
LBPR1	M18	057E, 1
LBPR2	M19	057F, 1
LBUFF	M20	0580, 96
LINBUF	B51	0247, 40
LMARGIN	B5	0052, 1
LOGCOL	B15	0063, 1
LOGMAP	B14	02B2, 4
LPENH	J5	0234, 1
LPENV	J6	0235, 1
MEMLO	A1	02E7, 2
MEMTOP	A2	02E5, 2
MLTTMP	B40	0066, 2
NEWCOL	B19	0061, 2
NEWROW	B19	0060, 1
NOCKSM	H15	003C, 1
NSIGN	M7	00EE, 1
OLDADR	B38	005E, 2
OLDCHR	B37	005D, 1
OLDCOL	B3	005B, 2
OLDROW	B3	005A, 1
OPNTMP	B40	0066, 2
PADDL0	J3	0270, 1
PADDL1	J3	0271, 1

PADDL2	J3	0272, 1
PADDL3	J3	0273, 1
PADDL4	J3	0274, 1
PADDL5	J3	0275, 1
PADDL6	J3	0276, 1
PADDL7	J3	0277, 1
PBPNT	F3	001D, 1
PBUFSZ	F2	001E, 1
PCOLR0	B7, P27	02C0, 1
PCOLR1	B7, P27	02C1, 1
PCOLR2	B7, P27	02C2, 1
PCOLR3	B7, P27	02C3, 1
PLYARG	M21	05E0, 6
POKMSK	P2	0010, 1
PRNBUF	F1	03C0, 40
PTEMP	F4	001F, 1
PTIMOT	F5	001C, 1
PTRIGO	J4	027C, 1
PTRIG1	J4	027D, 1
PTRIG2	J4	027E, 1
PTRIG3	J4	027F, 1
PTRIG4	J4	0280, 1
PTRIG5	J4	0281, 1
PTRIG6	J4	0282, 1
PTRIG7	J4	0283, 1
RADFLG	M24	00FB, 1
RAMLO	N1	0004, 3
RAMSIZ	A5	02E4, 1
RAMTOP	A4	006A, 1
RECVDN	H22	0039, 1
RMARGN	B6	0053, 1
ROWAC	B24	0070, 2
ROWCRS	B2	0054, 1
ROWINC	B21	0079, 1
RTCLOK	P3	0012, 3
SAVADR	B41	0068, 2
SAVIO	D7	0316, 1
SAVMSC	B36	0058, 2
SCRFLG	B9	02BB, 1
SDLSTH	P23	0231, 1
SDLSTL	P23	0230, 1
SDMCTL	P22	022F, 1
SHFAMT	B29	006F, 1
SHFLOK	E6	02BE, 1
SOUNDR	H10	0041, 1
SRTIMR	E8	022B, 1
SSFLAG	E4	02FF, 1
SSKCTL	H32	0232, 1
STACKP	H28	0318, 1
STATUS	H31	0030, 1
STICKO	J1, J7, J8	0278, 1
STICK1	J1, J7, J8	0279, 1

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555 - Rev. 0.2

STICK2	J1, J7, J8	027A, 1
STICK3	J1, J7, J8	027B, 1
STRIGO	J2, J7, J9	0284, 1
STRIG1	J2, J7, J9	0285, 1
STRIG2	J2, J7, J9	0286, 1
STRIG3	J2, J7, J9	0284, 4
SUBTMP	B48	029E, 1
SWPFLG	B44	007B, 1
TABMAP	B13	02A3, 15
TEMP	H23	023E, 1
TEMP1	D5	0312, 2
TEMP2	S4	0314, 1
TEMP3	D6	0315, 1
TIMER1	D3	030C, 2
TIMER2	D3	0310, 2
TIMFLG	D2, H25	0317, 1
TINDEX	B49	0293, 1
TMPCHR	B33	0050, 1
TMPCOL	B46	02B9, 2
TMPLBT	B47	02A1, 1
TMPROW	B46	02B8, 1
TMPX1	S5	029C, 1
TOADR	B40	0066, 2
TRAMSZ	N1	0004, 3
TSTAT	H29	0319, 1
TSTDAT	N2	0007, 1
TXTCOL	B4	0291, 2
TXTMSC	B52	0294, 2
TXTOLD	B53	0296, 6
TXTROW	B4	0290, 1
USAREA	R1	0080, 128
VBREAK	P16	0206, 2
VDSLST	P10	0200, 2
VIMIRQ	P13	0216, 2
VINTER	P15	0204, 2
VKEYBD	P17	0208, 2
VPRCED	P14	0202, 2
VSERIN	P18	020A, 2
VSEROC	P20	020E, 2
VSEROR	P19	020C, 2
VTIMR1	P21	0210, 2
VTIMR2	P21	0212, 2
VTIMR4	P21	0214, 2
VVBLKD	P12	0224, 2
VVBLKI	P11	0222, 2
WARMST	N13	0008, 1
WMODE	D14	0289, 1
XMTDON	H24	003A, 1

REPRODUCTION PROHIBITED WITHOUT PUBLICATIONS DEPT. APPROVAL
OPERATING SYSTEM, CO16555 - Rev.. 0.2

(ZBUFF	K2)	0043, 2
(ZDRV A	K3)	0045, 2
ZIOCB	G13	0020, 16
(ZSBA	K4)	0047, 2
ZTEMP1	M13	00F5, 2
ZTEMP3	M15	00F9, 2
ZTEMP4	M14	00F7, 2

MEMORY ADDRESS ORDERED LIST OF DATABASE VARIABLES

ADDRESS	VID	NAME
0000-0001	S7	LNZBS
0002-0003	N6	CASINI
0004-0006	N1	RAMLO, TRAMSZ
0007	N2	TSTDAT
0008	N13	WARMST
0009	N7	BOOT?
000A-000B	N12	DOSVEC
000C-000D	N3	DOSINI
000E-000F	A3	APPMHI
0010	P2	POKMSK
0011	E5	BRKKEY
0012-0014	P3	RTCLOK
0015-0016	C1	BUFADR
0017	G23	ICCOMT
0018-0019	S6	DSKFMS
001A-001B	L1	DSKUTL
001C	F5	PTIMOT
001D	F3	PBPNT
001E	F2	PBUFSZ
001F	F4	PTEMP
0020	G13, G14	ICHIDZ
0021	G15	ICDNOZ
0022	G16	ICCOMZ
0023	G17	ICOBAS
0024-0025	G18	ICBALZ, ICBAHZ
0026-0027	G19	ICPTLZ, ICPTHZ
0028-0029	G20	ICBLLZ, ICBLHZ
002A-002B	G21	ICAX1Z, ICAX2Z
002C-002F	G22, G24, G25	ICSPRZ
0030	H31	STATUS
0031	H13	CHKSUM
0032-0033	H16	BUFRLO, BUFFRHI
0034-0035	H17	BFENLO, BFENHI
0036	H11	CRETRY
0037	H12	DRETRY
0038	H21	BUFRFL
0039	H22	RECVDN
003A	H24	XMTDON
003B	H14	CHKSNT
003C	H15	NOCKSM
003D	D11	BPTR
003E	D13	FTYPE
003F	D12	FEOF
0040	D15	FREQ
0041	H10	SOUNDR
0042	P1	CRITIC
0043-0049	K1, K2, K3, K4, K5	ZBUFF, ZBUFP, ZDRVVA, ZSBA
004A	N4	CKEY
004B	N5	CASSBT

004C	B34	DSTAT
004D	B10	ATTRACT
004E	B12	DRKMSK
004F	B11	COLRSH
0050	B33	TMPCHR
0051	B30	HOLD1
0052	B5	LMARGN
0053	B6	RMARGN
0054-0056	B2	ROWCRS, COLCRS
0057	B35	DINDEX
0058-0059	B36	SAVMSC
005A-005C	B3	OLDDROW, OLDCOL
005D	B37	OLDCHR
005E-005F	B38	OLDDADR
0060-0062	B19	NEWROW, NEWCOL
0063	B15	LOGCOL
0064-0065	B39	ADRESS
0066-0067	B40	MLTTMP, OPNTMP, TOADR
0068-0069	B41	SAVADR/FRMADR
006A	A4	RAMTOP
006B	B42	BUFCNT
006C-006D	B43	BUFSTR
006E	B50	BITMSK
006F	B29	SHFAMT
0070-0073	B24	ROWAC, COLAC
0074-0075	B25	ENDPT
0076-0078	B22	DELTAR, DELTAC
0079-007A	B21	ROWINC, COLINC
007B	B44	SWPFLG
007C	E7	HOLDCH
007D	B45	INSDAT
007E-007F	B23	COUNTR
0080-00FF	SEE FLOATING POINT VARIABLE LIST AT END.	
0100-01FF	6502 STACK	
0200-0201	P10	VDSLST
0202-0203	P14	VPRCED
0204-0205	P15	VINTER
0206-0207	P16	VBREAK
0208-0209	P17	VKEYBD
020A-0208	P18	VSERIN
020C-020D	P19	VSEROR
020E-020F	P20	VSEROC
0210-0215	P21	VITMR1, VITMR2, VITMR4
0216-0217	P13	VIMIRQ
0218-0219	P4, H26	CDTMV1
021A-021B	P6	CDTMV2
021C-0221	P8	CDTMV3, CDTMV4, CDTMV5
0222-0223	P11	VVBLKI
0224-0225	P12	VVBLKD
0226-0227	P5, H27	CDTMA1
0228-0229	P7	CDTMA2

022A	P9	CDTMF3
022B	E8	SRTIMR
022C	P9	CDTMF4
022D	P28	INTEMP
022E	P9	CDTMF5
022F	P22	SDMCTL
0230-0231	P23	SDLSTL, SDLSTH
0232	H32	SSKCTL
0233	S14	unused
0234	J5	LPENH
0235	J6	LPENV
0236-0239	S8	unused
023A	H18	CDEVIC
023B	H19	CCOMND
023C-023D	H20	CAUX1, CAUX2
023E	H23	TEMP
023F	H30	ERRFLG
0240	N8	DFLAGS
0241	N9	DBSECT
0242-0243	N10	BOOTAD
0244	N11	COLDST
0245	S9	unused
0246	C2	DSKTIM
0247-026E	B51	LINBUF
026F	P24	GPRIOR
0270-0277	J3	PADDLO -- PADDL7
0278-027B	J1, J7, J8	STICKO -- STICK3
027C-0283	J4	PTRIGO -- PTRIG7
0284-0287	J2, J7, J9	STRIGO -- STRIG3
0288	S2	CSTAT
0289	D14	WMODE
028A	D10	BLIM
028B-028F	S10	unused
0290-0292	B4	TXTROW, TXTCOL
0293	B49	TINDEX
0294-0295	B52	TXTMSC
0296-029B	B53	TXTOLD
029C	S5	TMPX1
029D	B32	HOLD3
029E	B48	SUBTMP
029F	B31	HOLD2
02A0	B28	DMASK
02A1	B47	TMPLBT
02A2	B26	ESCFLG
02A3-02B1	B13	TABMAP
02B2-02B5	B14	LOGMAP
02B6	E9	INVFLG
02B7	B18	FILFLG
02BB-02BA	B46	TMPROW, TMPCOL
02BB	B9	SCRFLG
02BC	B20	HOLD4
02BD	S1	HOLD5
02BE	E6	SHFLOK
02BF	B16	BOTSCR

02C0-02C3	B7, P27	PCOLR0 -- PCOLR3
02C4-02C8	B8, P27	PCOLR0 -- PCOLR4
02C9-02E3	S11	unused
02E4	A5	RAMSIZ
02E5-02E6	A2	MEMTOP
02E7-02E8	A1	MEMLO
02E9	S15	unused
02EA-02ED	G11	DVSTAT
02EE-02EF	D1	CHBAUDL, CHBAUDH
02F0	B1	CRSINH
02F1	E2	KEYDEL
02F2	E1	CH1
02F3	P25	CHACT
02F4	P26	CHBAS
02F5-02F9	S12	unused
02FA	B55	CHAR
02FB	B54	ATACHR
02FC	E3	CH
02FD	B17	FILDAT
02FE	B27	DSPFLG
02FF	E4	SSFLAG
0300	H1, H2	DCB/DDEVIC
0301	H3	DUNIT
0302	H4	DCOMND
0303	H5	DSTATS
0304-0305	H6	DBUFLO, DBUFHI
0306	H7	DTIMLO
0307	S3	DUNUSE
0308-0309	H8	DBYTLO, DBYTHI
030A-030B	H9	DAUX1, DAUX2
030C-030D	D3	TIMER1
030E	D4	ADDCOR
030F	D8	CASFLOG
0310-0311	D3	TIMER2
0312-0313	D5	TEMP1
0314	S4	TEMP2
0315	D6	TEMP3
0316	D7	SAVIO
0317	D2, H25	TIMFLG
0318	H28	STACKP
0319	H29	TSTAT
031A-033F	G12	HATABS
0340	G1, G2	IOCB, ICHID
0341	G3	ICDNO
0342	G4	ICCOM
0343	G5	ICSTA
0344-0345	G6	ICBAL, ICBAH
0346-0347	G7	ICPTL, ICPTH
0348-0349	G8	ICBLL, ICBLH
034A-034B	G9	ICAX1, ICAX2
034C-034F	G10	ICSPR

0350-035F	G2-G10	(IOCB #1)
0360-036F	G2-G10	(IOCB #2)
0370-037F	G2-G10	(IOCB #3)
0380-038F	G2-G10	(IOCB #4)
0390-039F	G2-G10	(IOCB #5)
03A0-03AF	G2-G10	(IOCB #6)
03B0-03BF	G2-G10	(IOCB #7)
03C0-03E7	F1	PRNBUF
03E8-03FC	S13	unused
03FD-047F	D9	CASBUF
0480-06FF	R2	User Area

FLOATING POINT PACKAGE VARIABLES

00D4-00D9	M1	FRO
00DA-00DF	M2	FRE
00E0-00E5	M3	FR1
00E6-00EB	M4	FR2
00EC	M5	FRX
00ED	M6	EEXP
00EE	M7	NSIGN
00EF	M8	ESIGN
00FO	M9	FCHRFLG
00F1	M10	DIGRT
00F2	M11	CIX
00F3-00F4	M12	INBUFF
00F5-00F6	M13	ZTEMP1
00F7-00F8	M14	ZTEMP4
00F9-00FA	M15	ZTEMP3
00FB	M24	RADFLG/DEGFLG
00FC-00FD	M16	FLPTR
00FE-00FF	M17	FPTR2
057E	M18	LBPR1
057F	M19	LBPR2
0580-05FF	M20	LBFEND, LBUFF
05E0-05E5	M21	PLYARG
05E6-05EB	M22	FPSCR/FSCR
05EC-05F1	M23	FPSCR1/SCR1





